Wissenschaftliches Arbeiten mit LaTeX

Berndt Farwer (Hrsg.)

Preprint, 25. August 2005

Fachbereich Informatik
Universität Hamburg

Vorwort

Im wissenschaftlichen Betrieb sind schriftliche Darstellungen eines der wichtigsten Kommunikationsmittel überhaupt. Hierbei muss eine Vielzahl von Faktoren berücksichtigt werden. Zum Beispiel ist es in den Naturwissenschaften häufig erforderlich, mathematische Formeln in den Text zu integrieren. Hierzu sind spezielle Hilfsmittel erforderlich. In allen Bereichen des wissenschaftlichen Publizierens sind Diagramme zur Veranschaulichung zu erstellen und einzubinden. In den Musikwissenschaften werden Auszüge aus Partituren verwendet, so dass Notensatz erforderlich ist. Für den potentiellen Autoren stellt sich somit die schwierige Frage nach einem oder mehreren Hilfsmitteln bei der Erstellung von Seminararbeiten, Diplomarbeiten, Konferenzbeiträgen, o.ä.

Trotz (oder gerade wegen) der immer größer werdenden Komplexität von herkömmlichen (WYSIWIG-)Textsystemen, erfreut sich im wissenschftlichen Publizieren TEX und LATEX immer größerer Beliebtheit. Diese Textsatzsysteme machen den Autoren gleichzeitig zum Setzer, ohne jedoch eine Ausbildung im Layout oder Textsatz zu erfordern. Die strengen – in Dokumentenklassen und Stilvorlagen – vorgegebenen layouterischen Richtlinien sind meist vorgegeben, so daß der Autor sich um die Gestalt des Textes wenig zu kümmern braucht. Vielmehr kann er sich voll und ganz dem Inhalt widmen und das Layout dem Textsatzsystem überlassen.

Nicht zuletzt aus wirtschaflichen Gründen findet TEX bzw. LATEX auch im Verlagswesen immer mehr Anhänger. Wie bei jeder komplexen Software, ist aber auch beim Einsatz von TEX mit einigen Klippen und Fallgruben zu rechnen. In den hier zusammengestellten Artikeln werden einige dieser Schwierigkeiten aufgegriffen und praxisorientierte Lösungen angeboten.

Diese Sammlung von Beiträgen entstand im Sommersemester 2005 im Rahmen eines Proseminars am Fachbereich Informatik der Universität Hamburg. Ein Hauptaugen-

merk des Proseminars, nämlich das Abhalten eines Vortrages, kann naturgmäß in dieser schriftlichen Form nicht widergegeben werden. Die Beiträge erheben ferner nicht den Anspruch auf Vollständigkeit und es kann keine Gewähr für die Korrektheit der Darstellungen übernommen werden. In den meisten Fällen sind die Quellen sowie Hinweise auf weiterführende Literatur angegeben.

Ich habe mich bei der Zusammenstellung der Beiträge bemüht, einige gewisse „Vereinheitlichung der Stile“ zu erzielen. Dies ist in der begrentzten mir zur Verfügung stehenden Zeit sicherlich nur in geringem Maße gelungen.

Hamburg im Juni 2005 Berndt Farwer

Inhaltsverzeichnis

Kapitel 1

Logische Struktur von Texten

Lukman Iwan, Harald Brinkmann

1.1 Grundlagen

Harald Brinkmann

LaTeX bezeichnet ein kostenloses Textsatzsystem. Die Grundidee bei LaTeX ist, dass sich ein Autor ausschließlich mit dem Inhalt befassen sollte. Die äußere Form wird dabei von LaTeX nach allgemeinen Standards erstellt, ist aber in allen Einzelheiten anpassbar.

LaTeX kann man aus dem Internet herunterladen, die Windows-Version gibt es zum Beispiel unter:

http://prdownloads.sourceforge.net/miktex/small-miktex-2.4.1705.exe?download

Aber LaTeX ist grundsätzlich für jedes Betriebssystem im Internet zu haben. Auch am FB18 gibt es LaTeX auf CD. Wenn das Schicksal einen einsamen Reisenden zu einer bestimmten Zeit in Raum D-124 verschlägt, und dieser das Losungswort „Erstsemester CD“ sagt, wird ihm gegen einen kleinen Obolus eine CD mit allerlei nützlicher Software in die Hand gedrückt, u.a. auch o.g. LaTeX-Distribution, auf die sich meine Erfahrungen sowie die folgenden Erläuterungen beschränken.

Die Benutzung von LaTeX lässt sich in 4 Schritten beschreiben:

1. LaTeX installieren
2. Eine ASCII-Textdatei erstellen (`dateiname.tex`)
3. `latex dateiname` ausführen (erzeugt `dateiname.dvi`)
4. `dateiname.dvi` lässt sich mit einem Viewer (z.B. Yap) öffnen und drucken oder in `dateiname.pdf` konvertieren (mit dvipdfm)

Schritt 1 muss nur einmalig durchgeführt werden und ist zumindest nach meiner Erfahrung denkbar einfach: Installationsprogramm ausführen und den Anweisungen auf dem Bildschirm folgen ;).

Schritt 2 ist schon etwas anspruchsvoller, siehe Abschnitt 1.2.

Schritt 3 kann einfach in einer Shell eingegeben werden, falls `latex.exe` „erreichbar“ ist. Ansonsten findet man `latex.exe` unter `texmf\miktex\bin\`.

Schritt 4 ist optional. Hierzu lässt sich sagen, dass es allerlei Konverter zwischen `.dvi`, `.ps` und `.pdf` gibt.

1.2 Struktur eines Dokumentes

Hier soll etwas näher auf den Aufbau von Dokumenten im Allgemeinen und die Abbildung desselben auf die LaTeX-Eingabedatei `dateiname.tex` im Speziellen eingegangen werden. Zuerst sehen wir hier das obligatorische „Hallo Welt“-Beispiel:

```
\documentclass{article}   % Dokumentklasse festlegen
\begin{document}          % Beginn des Dokuments
  Hallo Welt.             % Inhalt
\end{document}            % Ende des Dokuments
```

Dies erzeugt ein Dokument mit „Hallo Welt“ als einzigem Inhalt. Grundsätzlich gelten folgende Konventionen:

Befehle beginnen mit \ und können einen Parameter übergeben bekommen:

```
\befehl {arg}
```

Oder auch mehrere:

`\befehl {arg 1} {arg 2} ...`

Im Gegensatz zu der geschweiften Klammer für notwendige Parameter wird `[]` für optionale Parameter verwandt. Das kann dann so aussehen:

`\befehl [arg 1] {arg 2} {arg 3}`

Groß- bzw. Kleinschreibung: `\befehl{}` $\neq$ `\Befehl{}`

Kommentare: Mit `%` kann man Kommentare in die Eingabedatei einfügen. Dies ist besonders hilfreich, wenn später noch Änderungen an einer bestehenden umfangreichen `.tex`-Datei vorgenommen werden müssen.

Leerzeichen: Entweder es ist an einer bestimmten Stelle eines, oder nicht. Mehrere aufeinanderfolgende werden wegrationalisiert.

Absätze erhält man, indem man eine Zeile freilässt.

Nun wollen wir ein wenig genauer hinsehen. Hierzu brauchen wir ein nicht so minimalistisches Beispiel, „Hallo Welt advanced“:

```
\documentclass[a4paper]{article}

\usepackage[german]{babel}

\date{\today}
\author{meine Wenigkeit}
\title{mein Werk}

\begin{document}
   \maketitle
   Hallo Welt.
\end{document}
```

Zuerst sehen wir uns die *Präambel* an. Das ist alles, was nicht zwischen `\begin{document}` und `\end{document}` steht. So ein Paar `begin`, `end` nennt man *Um-*

gebung. In der Präambel werden Einstellungen vorgenommen und in der Document-Umgebung steht der eigentliche Text.

Mit der ersten Zeile wird die Dokumentklasse auf Artikel gesetzt. Zusätzlich wird hier das Papierformat auf eine Deutsche Norm gesetzt. Die Dokumentklasse bestimmt grundlegende Attribute wie z.B. Seitenzahlen und Standardschriftart. Es gibt u.a. noch die Dokumentklassen Report, Book und Letter.

Dann wird mit `\usepackage` das *Paket* Babel eingebunden. Babel lokalisiert die LaTeX-Datei. Dabei werden Silbentrennung, Kapitel- und Indexüberschriften nach den jeweiligen Konventionen behandelt.

Desweiteren wird Datum, Autor und Titel bekanntgemacht, so dass - jetzt ein fließender Übergang zur Document-Umgebung - der Befehl `\maketitle` weiß was er zu tun hat. Mit `\maketitle` werden Datum, Autor und Titel hübsch auf eine einzelne Seite gebracht, und das an der Stelle, an der der Befehl steht, d.h. wenn jemand gerne seinen Lesern erst auf der letzten Seite verraten möchte, wie er sein Werk genannt hat, so würde er den `\maketitle`-Befehl direkt vor das `\end{document}` schreiben.

Um das Dokument baumartig zu strukturieren gibt es folgende Möglichkeiten:

`\chapter{Kapitelname}`

`\section{Abschnittsname}`

`\subsection{Unterabschnittsname}`

`\subsubsection{Unter-unterabschnittsname}`

Der chapter-Befehl steht nur bei der Dokumentklasse „Book“ zur Verfügung.

Um LaTeX mitzuteilen, was Kapitel, Anhang oder Vorwort ist, gibt es die drei Befehle `\frontmatter`, `\mainmatter` und `\backmatter`. Diese Befehle beeinflussen z.B. die Seitenzahlen (in der Frontmatter stehen Römische Zahlen).

Ein gutes Buch sollte folgende Features haben:

Frontmatter:

Deckblatt

Impressum

Titel

Inhaltsverzeichnis

Vorwort

Mainmatter:

Kapitel 1

⋮

Kapitel n

Backmatter:

Nachwort

Anhang A

⋮

Anhang Z

Abbildungsverzeichnis

Stichwortverzeichnis

1.3 LaTeX - Befehle

Lukman Iwan

Wir wir bereits vorher gesehen haben, werden Befehle in LaTeX werden immer mit einem \ begonnen (siehe Abschnitt 1.2), anschließend folgt der Befehlsname, dann Optionen und Argumente:

```
\befehlsname[Optionen]{Argument}
\befehlsname*[Optionen]{Argument}
```

Für viele Befehle gibt es jedoch noch die *-Variante. Sie haben gegenüber den normalen Befehl noch eine zusätzliche Besonderheit, die bei jeder *-Variante anders ist.

Hier sind einige Befehle zusammengefasst:

Ausgabe:	Quellcode:
Kursiver Text	`\textit{Kursiver Text}`
Fettgedruckter Text	`\textbf{Fettgedruckter Text}`
Unterstrichener Text	`\underline{Unterstrichener Text}`
`Schreibmaschinenschrift`	`\texttt{Schreibmaschinenschrift}`
*Her*vor*gehoben*	`\emph{Her\emph{vor}gehoben}`
small	`{\small small}`
large	`{\large large}`
huge	`{\huge huge}`
a b	`a b`
a b	`a  b`
a b	`a \ b`
a b	`a\,b`

`\,` verwendet man oft zwischen Zahl und Einheit, damit die Einheit näher an der Zahl steht und ein Zusammenhang deutlich wird. Außerdem sieht es auch schöner aus:

Ausgabe:	Quellcode:
13,37 cm	`13,37 cm`
13,37 cm	`13,37\,cm`

Eine neue Zeile erzwingt man mit `\\`. Mit `\\*` erzeugt man auch eine neue Zeile, aber bei diesem Zeilenumbruch wird kein Seitenumbruch zugelassen. Man beachte, dass die Zeile dort zuende ist, wo `\\` steht: die Zeile wird dort abgeschnitten, auch im Blocksatz. Damit LaTeX nur in eine neue Zeile springt und den Text als Blocksatz richtig darstellt, muss man den Befehl `\linebreak` verwenden.

Einen Absatz erzeugt man durch den Befehl `\par` oder einfach durch eine Leerzeile im Quelltext. Eine neue Seite kann man mit dem Befehl `\newpage` erzwingen. Verwendet man als Dokumentklasse `book`, dann möchte man z.B. für neue Kapitel auf der rechten Seite wieder anfangen. Der Befehl `\cleardoublepage` ist dafür richtig.

In LaTeX gibt es sogenannte Umgebungen, die mit `\begin{...}` begonnen werden und mit `\end{...}` wieder geschlossen werden. Umgebungen fassen bestimmte Teile als logische Blöcke zusammen.

Aufzählungen sind Umgebungen, die mehrere Aufzählungspunkte zusammenfassen. Dabei gibt es diese Aufzählungsarten: itemize, enumerate und description.

```
\begin{itemize}     % Normale Aufzählung
    \item 1. Punkt
    \item 2. Punkt
    \item 3. Punkt
    \item  ...
\end{itemize}

\begin{enumerate}   % Nummerierte Aufzählung
    \item Nummerierte Aufzählungen sind leicht.
    \item Jedes Item erzeugt einen neuen Punkt.
    \item % auch leere Items haben eine Nummer.
    \item ...
\end{enumerate}

\begin{description} % Beschreibende Aufzählung
    \item[Haus:] In einem Haus kann man wohnen.
    \item[Auto:] Mit einem Auto bewegt man sich fort.
    \item[Computer:] Sollte ein Informatiker kennen.
    \item ...
\end{description}
```

Beispiel für normale Aufzählung:

- 1. Punkt
- 2. Punkt
- 3. Punkt
- ...

Beispiel für nummerierte Aufzählung:

1. Nummerierte Aufzählungen sind leicht.
2. Jedes Item erzeugt einen neuen Punkt.

3.

4. ...

Beispiel für beschreibende Aufzählung:

Haus: In einem Haus kann man wohnen.

Auto: Mit einem Auto bewegt man sich fort.

Computer: Sollte ein Informatiker kennen.

...

Jede Umgebung, die man mit `\begin{umgebung}` öffnet, muss man mit `\end{umgebung}` wieder schließen.

Es gibt auch die Umgebungen `flushleft`, `flushright` und `center`, um Text links, rechts oder mittig auszurichten. Zeilenumbrüche innerhalb dieser Umgebungen kann man mit `\\` erzwingen. Als Standard ist Blocksatz eingestellt.

Dieser Text
steht in einer Umgebung
namens `flushleft`.

Dieser Text
steht in einer Umgebung
namens `flushright`.

Dieser Text
steht in einer Umgebung
namens `center`.

Mathematische Formeln werden auch in speziellen Mathe-Umgebungen geschrieben. Dabei werden u.a. die Variabeln kursiv gesetzt:

Ausgabe:	Quellcode:
$f(x) = 2ax + b$	`$f(x) = 2ax+b$`
f(x) = 2ax+b	`f(x) = 2ax+b`

Mehr zu dem Mathe-Kram gibt es im Kapitel „Mathematische Formeln in/mit LaTeX“.

1.4 Anpassen der Vorlagen

Da LaTeX in Amerika entwickelt wurde, sind die Standard Einstellungen auch auf den amerikanischen Raum zugeschnitten. Mit ein paar zusätzlichen Befehlen in der Preambel kann man die Vorlage an den lokalen Standard anpassen.

```
\documentclass[a4paper]{...}  % setzt das Papierformat auf DIN A4
                              % Standard ist US-Letter
\usepackage[ngerman]{babel}   % deutsche Silbentrennung
\usepackage[latin1]{inputenc} % Eingabe mit Umlauten
\usepackage[T1]{fontenc}      %
\usepackage{parskip}          % Absatzformatierung
```

1.5 Makros

Makros sind dafür da, um wiederholte Eingaben oder Formatierungen zu automatisieren. Man kann sie mit Prozeduren oder Funktionen vergleichen. Diese deklariert man in der Preambel und kann sie dann im Dokument wiederholt aufrufen.

Um ein neues Makro zu definieren oder auch zu überschreiben verwendet man folgende Befehle in der Preambel:

```
\newcommand{<name>}[param]{<befehle>}   % neues Makro definieren
\renewcommand{<name>}[param]{<befehle>} % Makro überschreiben
```

Beispiel:

```
\newcommand{\makrobspa}{Das hier wird unten redefiniert.}
\newcommand{\makrobspb}{Hier wurde \texttt{testb} aufgerufen
                        mit \underline{mehreren} Befehlen.}
\renewcommand{\makrobspa}{Dies hier ist nur ein Test!}
\newcommand{\makrobspc}[1]{Ein Makro mit einem Parameter
                           param=\textbf{#1}}
```

Im Beispiel wurde als erstes das Makro `\makrobspa` definiert, danach ein zweites Makro `\makrobspb`, welches auch weitere Befehle verwendet und anschließend wird das erste Makro redefiniert, d.h. überschrieben. Das dritte Makro `\makrobspc` verwendet noch die Option `[1]`, was bedeutet, dass dieses Makro beim Aufruf 1 Argument besitzt.

Im Dokument kann man die Makros dann über ihren Namen, hier also `\makrobspa`, `\makrobspb` bzw. `\makrobspc{...}` aufrufen. An diesen Stellen werden dann die Befehle ausgeführt:

Ausgabe:	Quellcode:
Dies hier ist nur ein Test!	`\makrobspa`
Hier wurde `testb` aufgerufen mit mehreren Befehlen.	`\makrobspb`
Ein Makro mit einem Parameter param=**Moin**	`\makrobspc{Moin}`

Manchmal will man das @-Zeichen als einen Teil des Makronamens verwenden. Dies ist von vorne herein nicht erlaubt. Dazu braucht man diese beiden Befehle:

```
\makeatletter
...             % hier ist @ als als Name erlaubt
\makeatother
```

Im Allgemeinen gilt für die Wahl von Makronamen, dass man Namen wählen sollte, die das Makro beschreiben. So sind Namen wie `\doit` oder `\asd` nicht sehr sinnvoll. Auch wenn Namen für Theoretiker Schall und Rauch sind, fördern Namen wie z.B. `\definition` die Lesbarkeit des Quelltextes für andere Personen erheblich. Auch für sich selbst sind solche Makronamen hilfreicher, um den Quelltext auch später irgendwann noch verstehen zu können, ohne erst nachzusehen, was das Makro macht.

Besonders für blinde Menschen ist dies wichtig, da sie nur den Quelltext vorgelesen bekommen. Wenn man z.B. in Definitionen die neu definierten Wörter durch Fettdruck hervorheben will, kann man den Befehl `\textbf{...}` verwenden. Für einen Blinden sagt das nicht viel, wenn er weiß, dass dieses Wort fett dargestellt werden soll. Besser wäre ein neues Makro mit dem Namen `\definition{...}` zu verwenden, was zwar den Text auch nur fett darstellt, aber jetzt wird auch der Sinn deutlich, warum dieses Wort fett dargestellt werden soll.

So kann man sich als Faustregel merken: Lieber ein Makro zuviel als eines zu wenig. Um auf das Beispiel mit den fettgedruckten neu definierten Wörter zurück zu kommen,

kann es sein, dass man später die neu definierten Wörter nicht mehr fett, sondern lieber kursiv dargestellt haben möchte. Hätte man einfach den Befehl `\textbf{...}` verwendet und macht ein Suchen und Ersetzen, wird man mit aller Wahrscheinlichkeit auch andere fettgedruckten Wörter nun als neu definierten Wörter verstehen. Wenn man sich dafür ein eigenes Makro geschrieben, um die neu definierten Wörter hervorzuheben, kann man dieses Makro an einer Stelle schnell ändern und spart sich dadurch viel Arbeit.

1.6 Vergleich LaTeX - Word

In diesem Abschnitt wollen wir LaTeX mit dem bekannten Schreibprogramm Word von Microsoft vergleichen.

Die Vorteile von LaTeX sind folgende:

- LaTeX ist kostenlos
- LaTeX gibt es für alle Plattformen. Die tex-Dateien sind normale Textdateien, die auf allen Plattformen erstellt, gelesen und ohne Probleme verändert werden können.
- Versionsunabhängig: Man kann alte tex-Dateien auch mit neuem LaTeX weiterverwenden
- klare, eindeutige Formatierungen
- Automatisches Layout mit Möglichkeit manuell Veränderungen vorzunehmen
- Index, Bibliographie und Abbildungsverzeichnis werden mit entsprechenden Befehlen automatisch erzeugt und angezeit.
- Silbentrennung: Je nach verwendete Sprache kann man die nötige Silbentrennung laden und LaTeX trennt die Worter an den richtigen Stellen.
- Vernünftige Eingabemöglichkeiten für mathematische Formeln
- Universelle Lesbarkeit: die tex-Dateien sind ganz normale Textdateien, die von jedem geöffnet und gelesen werden können. Vor allem blinde Menschen haben keine Schwierigkeiten den Inhalt sich mit einer Software vorlesen zu lassen. Zudem haben

sie die Möglichkeit, den Sinn besser zu verstehen, da sie auch die Befehle vorgelesen bekommen, z.B. `\emph` (Emphasize) um etwas hervorzuheben, statt einfach `\textbf`, was einen Text einfach fett setzt.

Vorteile von Word:

- Es ist ein WYSIWYG-Texteditor
- Also für Anfänger einfach zu bedienen
- Weit verbreitet, d.h. Word ist auf vielen Computern vorhanden, so dass man es dort benutzen kann

Nachteile von Word:

- Speichert persönliche Daten: in jedem Dokument wird z.B. automatisch der Name als Autor eingetragen, auf dem das Produkt registriert wurde
- Großer Platzbedarf: mehr als 24000 Zeichen für ein einziges leeres Blatt
- Unerwünschte Nebeneffekte: Word korregiert während der Eingabe angebliche Tippfehler, auch solche die keine Tippfehler sind, so dass man sich oft darüber ärgert, warum Word das nicht so lassen kann, wie man es selbst eingetippt hat
- Rechtschreibprüfung und „intelligente“ Grammatikprüfung: Jeder, der die Grammatikprüfung schonmal getestet hat, weiß, dass er damit mehr Arbeit bekommt als dass ihm geholfen wird.

Kapitel 2

Präsentationen mit LaTeX

Anne-Kathrin Peters, Berndt Farwer

Zum Erstellen wissenschaftlicher Texte gehört in der Regel auch ein Vortrag der Inhalte, der meist in Form einer (Folien-)Präsentation durchgeführt wird. In diesem Kapitel zeigen wir, wie LaTeX zur Erstellung einer solchen Präsentation verwendet werden kann. Bevor wir uns in Abschnitt 2.2 mit den technischen Anforderungen LaTeXs beschäftigen, wollen wir jedoch zunächst in Abschnitt 2.1 den generellen Aufbau einer Folienpräsentation diskutieren und einige wichtige Grundbegriffe erklären. Dabei gehen wir auch auf die Ergonomie des Foliendesigns.

2.1 Allgemeines zu Präsentationen

2.1.1 Einleitung

> Das Leben ist eine einzige Selbstdarstellung. Das glauben Sie nicht?
>
> Dann führen Sie sich doch einmal wichtige Stationen Ihres Lebens vor Augen!
>
> Beim Vorstellungsgespräch, vor einem wichtigen Abschluss beim Kunden, beim letzten Vortrag vor einem größeren Publikum - und auch bei Ihrem letzten heißen Rendezvous: Haben Sie nicht versucht, Ihre Fähigkeiten optimal darzustellen? Sich als den nettesten, intelligentesten, charmantesten partnerschaftlichsten Kerl (Pardon meine Damen) zu präsentieren? [Fri03]

Die in diesem Abschnitt beispielhaft präsentierten Situationen des Lebens haben eines gemeinsam:

Es kommt immer darauf an, sich selbst ins Licht zu rücken, bzw. darzustellen.

Dies sollte auch im Folgenden bedacht werden, wenn ich einige Hinweise gebe, wie man geschickt wissenschaftliche Vorträge, z.B. in Firmen und Universitäten, halten kann.

Ein aktuell sehr beliebtes Medium ist dabei die Präsentation mittels Beamer. Das grazile Auf- und Abdecken mehrerer übereinander gelegter Folien auf dem Overheadprojektor wurde abgelöst durch entsprechende Software. Auch Packages in Latex bieten die Möglichkeit einer solchen Beamer-Präsentation.

2.1.2 Wichtige Begriffe

Folgende Begriffe klären den Zusammenhang zwischen der beamergestützen Präsentation und dem Vortragen mit Hilfe des Overheadprojektors:

Präsentation Ist eine mediengestützte Vorstellung von

- Ergebnissen
- Ideen und Konzepten
- Produkten

Medien sind Hilfsmittel, um Informationen zu einem Publikum zu transportieren

Slide ist der Begriff für eine Folie. In der Beamer Umgebung wird dafür der Begriff “Frame“ benutzt.

Overlay nennt man übereinandergelegte Folien mit denen man Inhalt ergänzt.

Builds ist ein in Zeiten von Computerpräsentation entstandener Begriff für Overlays.

2.1.3 Medien

Neben der Möglichkeit der Präsentation mit dem Beamer oder Overheadprojektor gibt es andere Medien, die man in seinen Vortrag einbauen kann. Ein Wechsel des Mediums ist eine dem Publikum willkommene Abwechslung. Der Vortrag wird lebhafter, je anschaulicher und kreativer er ist.

Tafel ist ein schönes Spontanmedium, um Sachverhalte zu skizzieren.
Argumente, Ideen o.ä. können durch den Vortragenden, oder mit dem Publikum an der Tafel zusammengetragen werden

Flip Chart ist eine Art "großer Schreibblock", der auf 3 Beinen steht. Ein Flip Chart ist in vielen Seminarräumen in Firmen vorhanden.
Es eignet sich ebenfalls gut als Spontanmedium: Einzelne Seiten können vorbereitet werden oder während der Präsentation mit oder von dem Publikum entworfen werden. Nachteil: Wird das Flip Chart während der Präsentation entworfen, wird die Schrift leicht unleserlich und schief (Tip: vorher dünn, mit Bleistift Hilfslinien ziehen). Der Vortragende steht ungünstig (Tip: man sollte versuchen, sich nicht komplett mit dem Rücken zum Publikum zu stellen. Das lässt sich vorher üben!)

Folien, Slides werden bei Präsentationen mit dem Beamer oder Overheadprojektor gebraucht

Pin Chart oder auch Plakat. Gut im Raum plaziert, wird es zu einem ständigen optischen Blickfang, der sich nachhaltig einprägt. Voraussetzung ist eine sorgfältige Gestaltung. Auch die Gliederung kann in Form eines Pin Charts für alle sichtbar in den Raum gehängt werden.

Zeitung, Bücher Zum Aufgreifen aktueller Ereignisse. Weckt das Interesse der Zuhörer.

Demonstrations-Objekt Das Objekt ist greifbar und anziehend. Es weckt die Neugier und den Spieltrieb der Zuhörer. Vorsicht: Nur bei inhaltlichem Bezug benutzen! Bezug herstellen, Objekt *aufwertend* präsentieren. Das Objekt sollte vor der Präsentation sichtbar und dekorativ an eine unauffällige Stelle gestellt werden. Publikum muss merken: Vorstellende identifiziert sich mit dem Objekt. z. B. Objekte mit optischer oder akustischer Signalwirkung:

- Sonnenbrille: Vorstellbar wäre, dass ein Mitarbeiter seine Firma vorstellt, um Kunden zu gewinnen. "Warum sollten Sie sich für unser Unternehmen entscheiden?" Weil unser Unternehmen eine strahlende Zukunft hat)
- Toilettenpapier: Zum Erklären des Konstrukts "Liste", bei denen man immer nur ein Element entfernen kann. Wieviele weitere Elemente die Liste bzw. das Toilettenpapier noch enthält, ist nicht absehbar.

Film Er sollte das Präsentationsziel decken. Präsentation:
"Häppchenweise", also mit Unterbrechungen zur Auswertung bzw. Diskussion, oder im Ganzen mit Diskussion und Auswertung am Schluß.

2.1.4 Vorüberlegungen

Um eine interessante Präsentation durchzuführen sollte sich der Vortragende einige Fragen stellen: In was für einem Raum wird der Vortrag gehalten? Zu welcher Tageszeit? Wieviel Zeit habe ich? Was sind die wichtigsten Punkte? Wie kann ich ein Problembewusstsein bei meinen Zuhörern entwickeln? Welche Informationen benötige ich? Warum wird diese Präsentation geführt? Welches Ziel möchte ich erreichen?

Das Zielpublikum sollte analysiert werden. Auf welche Vorkenntnisse kann ich bauen? Wofür interessieren sich die Zuhörer?

Nach einer Informationssammlung und -Strukturierung, sollte der Vortragende versuchen die Interessen der Zuhörer mit seinem Wissen zu vereinbaren.

Eine Gliederung, bestehend aus einer Einleitung, einem Hauptteil und einem Schluss sollte erstellt und mit Inhalten gefüllt werden.

Hilfreich ist es, sich bereits in den Vorüberlegungen Antworten auf häuftig gestellte Fragen zu überlegen.

2.1.5 Präsentationsaufbau

In diesem Abschnitt möchte ich einen möglichen Präsentationsaufbau vorschlagen, wobei nicht alle Punkte berücksichtigt werden müssen:

1. **Einführung:** Dazu gehört die Vorstellung der eigenen Person und ein Aufhänger um zum Thema zu führen: Die Einleitung sollte prägnant sein und Interesse für das Thema wecken. Der Vortragende sollte die Zuschauer mit einem Problem konfrontieren, oder sogar provozieren. Die ersten Sätze sollten überlegt sein. Gut geeignete "Starter" sind:

    - Eine positive Meldung: "Letzte Woche habe ich gelesen, dass ... "

- Ein aktuelles persönliches Erlebnis: “Als ich eben über den Flur ging, bemerkte ich . . . “
- Eine provozierende Frage: “Wie gesund ist ein Unternehmen, wenn die Mitarbeiter ständig krank sind?“
- Ein ehrliches Kompliment “Ich freue mich, vor einem Publikum zu stehen,das (Leistung XY) erreicht hat.“
- Ein passender, nicht schon allseits bekannter Witz

Die Ziele des Referats sollten in der Einleitung deutlich werden und der Umfang des Themas sollte abgegrenzt werden.

2. **Überblick:** Die Gliederung sollte vorgestellt werden.

3. Formulierung der genauen **Themen- und Fragestellung**: Der Vortragende sollte die Terminologie und sein methodisches Vorgehen zur Ermittlung des Ergebnisses kurz erläutern.

4. **Durchführung:** Die Themen und Fragestellungen werden bearbeitet. Dazu werden Texte analysiert, Experimente durchgeführt, bzw. Informationen vermittelt. Daraus folgen Ergebnisse, bzw. Antwort zur Themen- und Fragestellung.
   Die Resultate werden diskutiert.

5. **Zusammenfassung:** Entscheidende Ergebnisse werden sturkturiert auf den Punkt gebracht. Dabei sollte immer wieder auf die Vortragsgliederung und die am Anfang gesetzten Ziele eingegangen werden.

6. **Anwendung, Bedeutung:** Es wird ein Praxisbezug hergestellt und dadurch auf die Bedeutung des Ergebnisses eingegangen. Der Vortragende sollte kontrollieren, ob die Inhalte verstanden wurden.

7. Der Vortragende kann eine **Diskussion** durch Provokation des Publikums, inhaltliche Fragen und Bemerkungen einleiten. Darin stellt er Widersprüche und andere Ansichten heraus. Anschließend wird der Vortrag methodisch reflektiert: “Wurde das Thema angemessen behandelt?“

8. **Schluss:** Die Kernaussagen werden hervorgehoben. Auch die Schlussformulierung sollte überlegt sein. Gut geeignete Möglichkeiten:

- "Mein Appell: Stimmen Sie ... zu!"
- "Der nächste Schritt sollte sein: ... "

2.1.6 Verkehrsregeln für Folien

1. Schrift:
   a) Schriftart: einheitlich (max. 2 Schriftarten, -größen), serifenlos, Trennung vermeiden
   b) Schriftgrößen:
      i. Fußnoten: 16-18 pt
      ii. Normaler Text: 20-24 pt
      iii. Überschriften: 32-48 (auch größer)
   c) Stichworte, statt ganzer Sätze
2. Folienaufbau:
   a) Rechtschreibfehler, ungünstige Blattaufteilung, Kontrastarmut vermeiden
   b) **einheitliches** Format, besser Querformat
   c) Foliennummern zur Orientierung in Kopf-, oder Fußzeile. Optional: Titel, Logo, Datum
   d) bei Zahlenmaterial und Diagrammen Untertitel mit Maßstab nicht vergessen
   e) interessante, aussagekräftige Überschrift
   f) max. 3 unterschiedliche, kontrastreiche Farben: rot wirkt aggressiv- daher nur für Markierung (sparsam, zielgerichtet)
   g) Ein dunkler Hintergrund (mit heller Schrift) ist optisch eindrucksvoll. Farbverläufe und Marmorierungen im Hintergrund verringern die Lesbarkeit.
   h) Platz lassen für Ausführungen und Ergänzungen
   i) Bilder, grafische Elemente und Schaubilder zur Anschaulichkeit einbinden
   j) Elemente schrittweise **ohne** Klang einfügen
   k) **Corporate Identity:** Firmenlogo, von der Firma bevorzugte Schriftart und -farbe benutzen

2.1.7 Vortragen der Präsentation

Vor der Präsentation gibt es einiges zu organisieren: Die Projektion bzw. Präsentation sollte von jedem Platz zu jedem Zeitpunkt gut sichtbar und hörbar sein. Der Vortragende sollte mit der Bedienung der Medien vertraut sein.

Es sollten nicht mehr als 3 Slides pro Präsentationsminute gezeigt werden. Die Präsentation sollte dicht an Thema, Ziel und Gliederung ausgerichtet sein. Um die Lebhaftigkeit des Vortrags zu erhöhen, kann man die Zuhörer an der Lösung offener Probleme beteiligen.

Der Vortragende sollte sich mit seiner Präsentation in den Vordergrund stellen: Er setzt die Akzente, er hebt Schwerpunkte hervor. Aktion macht die Präsentation lebendig. Wechseln bzw. Variation der Medien und Einbeziehen des Publikums begünstigt das.

Lebendige und spannende Beispiele, rhetorische und provokative Fragen werden vom Publikum dankend angenommen.

Der Vortragende sollte das Problembewusstsein der Zuhörer sicher stellen. Beim Vortragen gilt folgendes Prinzip: “ Vom Einfachen zum Schweren und vom Bekannten zum Unbekannten.“

Beim Präsentieren sollte sich der Vortragende nicht mit dem Rücken zum Publikum drehen. Die (Funk)maus nehmen Rechtshänder am Besten in die linke Hand. So ist die rechte frei für Gestik. Der Vortragende sollte einen lockeren und sicheren Eindruck machen und den Zuhörer mit seiner Gestik und Mimik mitreißen. Es sollte laut, deutlich, langsam und frei in kurzen Formulierungen gesprochen werden.

Tip: Wenn man zu den Zuhörern in der letzten Reihe spricht, ist es normalerweise laut genug.

Um Akzente zu setzen und Eintönigkeit zu vermeiden sollte die Stimme variiert werden. Der Vortragende sollte darauf achten, jeden anzusehen.

2.1.8 Bei Fragen danach

Der Fragende sollte ernst genommen werden. Dazu gehört, dass er ausreden kann und der Vortragende antwortet auch bei unsinnigen Fragen geduldig. Bezieht sich die Frage auf eine Folie, so wird diese nochmal gezeigt. Fragen sind keine persönliche Kritik. Nicht in

eine generelle Verteidigungshaltung verfallen! Einzelne Fragen, die nicht von allgemeinem Interesse sind, können später in einer kleineren Gruppe besprochen werden. Wenn die Frage unklar ist, bittet man den Fragenden, sie noch einmal zu wiederholen eventuell mit einer anderen Formulierung. Nicht auf Gedeih und Verderb eine Antwort konstruieren! Entweder man stellt die Frage in die Runde oder man gibt offen zu, dass man sie nicht beantworten kann

2.1.9 Schlusswort

Alle diese Tips lassen sich nicht auf einmal umsetzen. Jedoch lässt sich das Präsentieren üben. Man gewinnt Routine und hat so die Möglichkeit auf kleine Dinge zu achten und kreativ zu sein. Als einflussreicher Mitarbeiter einer Firma wird man, häufiger als man denkt, in eine Situation kommen, zu präsentieren. Wenn man sich vorher ausprobiert hat, einen lockeren sicheren Eindruck macht, kann man die Zuschauer üeberzeugen und mitreißen. So hat man im Leben, nicht nur in der Firma, einen entscheidenden Vorteil!

2.2 Präsentationen mit LaTeX

2.2.1 Die `beamer`-Dokumentenklasse

`beamer`-Basics

Wir wollen an dieser Stelle etwas genauer auf eine relativ junge LaTeX-Dokumentenklasse eingehen, das in der Lage ist, direkt PDF-Präsentationen zu erstelen. Dabei wird `pdflatex` an Stelle von `latex` verwendet.

Die Darstellung in diesem Abschnitt will und kann nicht alle Möglichkeiten dieser Klasse aufzeigen, sondern soll vielmehr den Leser in die Lage versetzen, möglichst schnell gut aussehende Präsentationen zu erstellen.

Trotzdem stellt sich natürlich die Frage, warum wir gerade die `beamer`-Dokumentenklasse [Tan04] ausgewählt haben? Die Frage lässt sich leicht mit drei Schlagwörtern beantworten. Es ist **einfach**, **elegant** und hinreichend gut **etabliert**.

Nun sind diese Schlagwörter für sich genommen nicht sehr aussagekräftig. Es bedarf also einer kurzen Erläuterung, was damit gemeint ist:

- Für die Erstellung von PDF-basierten `beamer`-Präsentationen benötigt man nur sehr wenige neue Befehle, die über die Erstellung von „normalen“ Texten mit LaTeX hinaus gehen. Überdies werden – zum Beispiel für Aufzählungen und Listen – in der Regel Standard-LaTeX-Konstrukte verwendet, so dass eine Übernahme von Teilen bestehender Texte einfach möglich wird.
- Die Eleganz mit LaTeX erstellter Texte wird durch das `beamer`-Package auch auf Präsentationen übertragen. Ein gewisser Minimalismus lässt auf den erstellten Folien das Wesentliche in den Vordergrund treten, ohne dabei Elemente einzubüßen, die der Übersicht über den Vortrag als Ganzem ermöglichen.
- Die Klasse wird – im Gegensatz zu einigen der anderen erwähnten Pakete – aktuell weiterentwickelt und verwendet moderne Konzepte.

Die Dokumentenstruktur folgt für die `beamer`-Dokumentenklasse immer dem folgenden Schema, wobei in den eckigen Klammern noch Optionen stehen können.

```
\documentclass[]{beamer}
\usepackage{pgf}
\usepackage{graphicx}
\usepackage{url}
\usepackage[german]{babel}
\usepackage[utf8]{inputenc}
\usepackage[T1]{fontenc}

    ⋮

\begin{document}

    ⋮

\end{document}
```

Folien werden folgendermaßen programmiert:

```
\begin{frame}
\frametitle{Titel der Folie}

  Text der Folie

    ⋮

\end{frame}
```

Ein wichtigste Struktur- und Gliederugngsmittel für Präsentationen sind *Overlays*, also Inhalte, die sich nach und nach auf der Folie ergänzen. Am einfachsten lassen sich Overlays für Listen erstellen, aber auch weiterführende Overlays, z.B. von Grafiken lassen sich mit dem `beamer`-Package recht intuitiv erstellen.

Overlays bei Listen lassen sich nach dem folgenden Schema erstellen:

```
\begin{itemize}
  \item<1-> First point.
  \item<2-> Second point.
  \item<3-> Third point.
\end{itemize}
```

In spitze Klammern eingefasst ist hier jeweils der Bereich der Overlays, auf denen der Inhalt erscheinen soll. Einige Beispiele hierfür sind

`<1>` nur auf dem ersten Overlay

`<1->` auf allen Overlays ab dem ersten Overlay

`<1-3>` auf dem ersten bis dritten Overlays ab dem ersten Overlay

`<1,3-5>` auf dem ersten sowie dritten bis fünften Overlay

Wird statt `\begin{\itemize}` der Befehl `\begin{\itemize}<+->` verwendet, so werden die einzelnen Spiegelpunkte (`\item`) nacheinander eingeblendet, ohne dass die explizit angegeben werden muss. Verwendet man `\begin{frame}[<+->]` So wird jeder Spiegelpunkt und jede Umgebung nacheinander aufgebaut.

Weitere Overlay-Formen sind nicht an Listen oder Aufzählungen gebunden. Die Angabe der Overlays funktioniert aber ähnlich. Da Folieninhalte standardmäßig vertikal zentriert werden, wird hier der Befehl `\uncover` verwendet, der bereits ab dem ersten Overlay genug Platz für die später aufzudeckenden Teile reserviert. Ohne diese Maßnahme würde ein vertikales Ruckeln beim weiterschalten entstehen. Dies ist beim letzten Overlay (`\only<5>{End.}`) zu sehen:

```
\only<1-2>{First point.\\}
\uncover<3->{Second point.\\}
\uncover<1,3>{Third point.\\}
\only<5>{End.}
```

Zusaätzlich zum gerade besprochenen vertikalen Ruckeln, ist in diesem Beispiel noch ein unschönes horizontales Ruckeln festzustellen, was bei dem Befehl `\uncover<3->` ausgelöst wird. Es stammt daher, dass der Zeilenumbruch vor dieser Zeile von LaTeXwie ein Leerzeichen interpretiert wird. Um das Ruckeln zu vermeiden, sollte besser ein `%` am Ende der vorigen Zeile verwenden:

```
\only<1-2>{First point.\\}%
\uncover<3->{Second point.\\}%
\uncover<1,3>{Third point.\\}%
\uncover<5>{End.}%
```

Erzeugen der Präsentation

Das Erzeugen der Präsentation erfolgt mit `pdflatex`, z.B. über den Kommandozeilenbefehl

```
> pdflatex datei.tex
```

- Dabei wird eine PDF-Datei mit dem Namen `datei.pdf` erzeugt, die mit einem beliebigem PDF-Viewer betrachtet werden kann.

- Der Adobe Reader unterstützt auch Transitions-Effekte und einen „Full-Screen"-Modus.
- Alternativ zur Verwendung von `pdflatex` kann auch folgendermaßen vorgegangen werden:

```
> latex datei.tex
> dvips -o datei.ps datei.dvi
> ps2pdf datei.ps
```

Grafikformate mit Unterstützung durch `beamer` sind:[1]

- standardmäßig **PDF**, **JPG**, **JPEG** und **PNG**
- **PS** und **EPS** nur bei Verwendung von `latex` und `dvips`
- **MPS** (MetaPost) sind spezielle EPS-Dateien, werden aber von `pdflatex` unterstützt
- ebenso **MMP** (Multi-MetaPost)

Beispiel-Vorlagen: DEFAULT wird verwendet durch `\usetheme{default}`

Es handelt sich um eine schlichte Vorlage **ohne Navigationsleiste** und **ohne Kopf- und Fußzeilen**. Beispiele für diese Vorlage sind in Abb. 2.1 zu sehen.

Beispiel-Vorlagen: MADRID wird verwendet durch `\usetheme{madrid}` bzw. `\usetheme[`*<options>*

Als Option ist beispielsweise `secheader` möglich, wodurch `section`- und `subsection`-Titel als Kopfzeile angezeigt werden. Abb. 2.1 zeigt Beispiele für diese Vorlage.

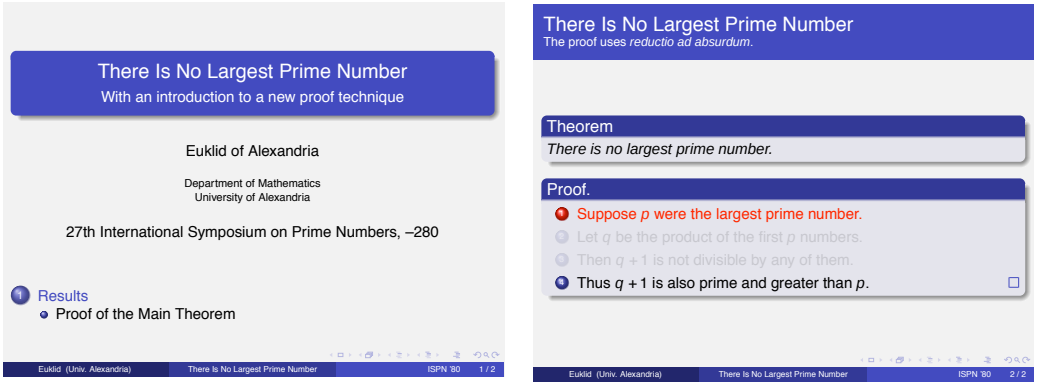
Verändern der Gestaltung

Mit der `beamer`-Dokumentenklasse ist man aber nicht nur auf die vordefinierten Vorlagen festgelegt. Man hat vielfältige Gestaltungsmöglichkeiten, selbst wenn man eine der Standardvorlagen verwendet.

[1] Es ist das Paket `graphics` oder `pgf` erforderich.

(a) beispiel1 (b) beispiel2

Abbildung 2.1: Folienvorlage `default`



(a) beispiel1 (b) beispiel2

Abbildung 2.2: Folienvorlage `madrid`

Optionen der Gestaltung von Präsentationen sind zum Beispiel:

- Verwendete (individuelle) Farben bzw. Farbthemen für die Titelgestaltung und den Folienhintergrund
- *inner* und *outer styles*
- Themenvorlagen
- Verwendung einer Grafik als Logo auf allen Folien
- Icons z.B. für die Literaturangaben

- Variationen der Navigationsleiste sowie Kopf- und Fußleisten
- Variationen verwendeter Boxen und Schatten
- Überblend-Effekte (Transitions)
- Draft-Modus
- u.v.m.

Der Draft-Modus erlaubt es, die Folien ohne Einbindung von Grafiken zu erstellen, was die Kompilierzeit erheblich senken kann. Hierbei werden die für die Grafiken benötigten Bereiche einfach durch Kästen markiert, so dass die Einteilung der Folien unberührt bleibt.

Text, Präsentation und Handout

Meist ist eine Präsentation nur ein Teil einer wissenschaftllichen Arbeit und wird ergänzt durch einen Artikel und ggf. ein Handout. Um die Erstellung all dieser Medienformen zu erleichtern, bietet die `beamer`-Dokumentenklasse Möglichkeiten der integrierten Erstellung all dieser Teile.

Eine Auswahl an Konstrukten hierfür ist wird in diesem Abschnitt kurz eingeführt.

Modi für unterschiedliche Medien lassen sich innerhalb der LaTeX-Source umschalten und optional mit Parametern versehen. Einige Beispiele hierfür sind:

Präsentation: `\mode<presentation>`

```
\mode<presentation>{\usetheme{Berlin}}
```

Handouts: `\mode<handout>`

```
{\beamertemplatesolidbackgroundcolor{black!5}}
```

Text/Artikel: `\mode<article>{\usepackage{fullpage}}`

2.2.2 Weitere Möglichkeiten der Erstellung von Präsentationnen mit LaTeX

An dieser Stelle seien nur kurz einige weitere Möglichkeiten der Erstellung von Folien mit LaTeX erwähnt. Die genaue Syntax wird hier nicht diskutiert. statt dessen stellen wir beispielhaft typische Ergebnisse der Styles vor.

`slides` und der `seminar`-Style

- `slides` ist standardmäßig in LaTeX enthalten.
- `seminar` is eine Erweiterung von `slides`.

```
\documentclass[a4,landscape]{slides}
```

⋮

```
\begin{slide}
Linear Logic operators ...
\begin{enumerate}
  \item The so-called ...
  \item The \defi{mul ...
  \item Linear \defi{ ...
  \item The \defi{exp ...
\end{enumerate}
\end{slide}
```

Linear Logic operators are divided into four groups:

1. The so-called *additive* connectives basically wo way as their classical counterparts.
2. The *multiplicative* operators are used to acc sources.
3. Linear *Negation*.
4. The *exponentials* are modalities that have to b to regain the power of classical and intuitionisti

Obiges Code-Fragment ergibt eine Folie mit einem Aussehen, wie durch das rechts daneben abgebildete Folien-Fragment angedeutet.

`prosper`

Die `prosper`-Klasse [Wil00] hat einige Eigenheiten:

- Sie arbeitet **nicht** mit `pdflatex` zusammen!
- Die Anzahl der Overlays einer Folie muss vorher bekannt sein.

- Es wird PostScript erzeugt, das dann in PDF konvertiert werden muss
- Grafiken können nur im PS-Format eingebunden werden.
- Es gibt eine Auswahl von etwa einem Dutzend Vorlagen.

Beispiele finden sich in Abb. 2.3.

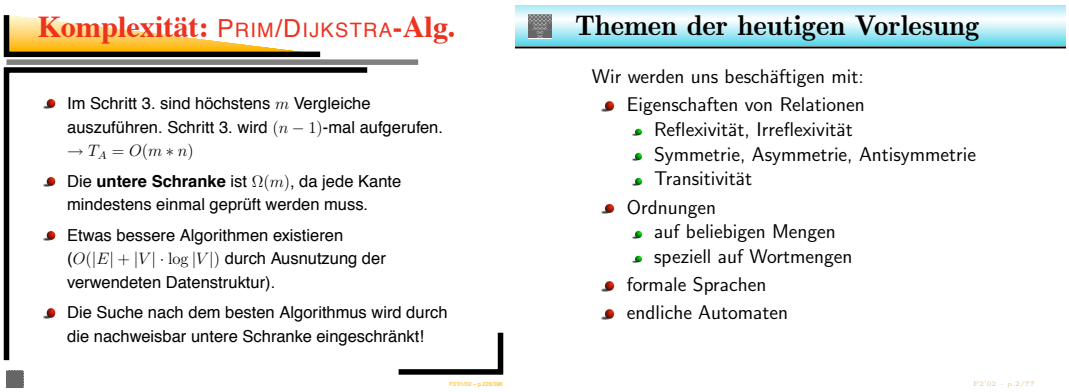


Abbildung 2.3: Beispiele für `prosper`-Folien

`utopia`

Das `utopia` Presentation Bundle [Gou00] ist

- kommerziell
- PostScript-basiert
- Post-Prozessor
- mit einfachen, mächtigen Befehlen ausgestattet
- mit simplen aber schönen Vorlagen bestückt
- für schnelle Erstellung (z.B. durch `\blob`, `\Blob`, `\bblob`) von Folien optimiert.

Beispiele sind in Abb. 2.4 zu finden.

Abbildung 2.4: Beispiele für `utopia`-Folien

Kapitel 3

Grafiken in Texten

Kristof Hamann, André Monsees

Neben reinem Text spielen auch Grafiken in wissenschaftlichen Arbeiten eine Rolle. Zu Beginn dieses Kapitels befindet sich eine kurze Einführung in das Thema Grafiken in Texten. In Abschnitt 3.2 wird darauf eingegangen, wie Grafiken in LaTeX-Dokumente eingefügt, platziert und formatiert werden. In Abschnitt 3.3 werden einige Pakete vorgestellt, mit denen direkt aus LaTeX Grafiken erstellt werden können.

3.1 Einführung

Mit Wörtern können wir zwar nahezu alles ausdrücken, doch es kann hilfreich oder gar notwendig sein, einen fließenden Text durch Grafiken zu ergänzen. Sei es um einen komplizierten Sachverhalt übersichtlich darzustellen (wie es beispielsweise durch Abbildung 3.1 auf Seite 34 gemacht wird), Fotos, Diagramme oder mathematische Objekte wie Graphen oder Automaten abzubilden – sie alle unterscheiden sich in ihren Eigenschaften stark vom Rest des Textes und stellen somit als Fremdkörper besondere Ansprüche an die Textverarbeitung.

Eine Grafik sollte möglichst gut in den fließenden Text integriert werden. Deshalb ist es wichtig, dass die Grafik dort eingefügt wird, wo sie inhaltlich zum Text passt. In jedem Fall ist eine möglichst ausführliche Beschreibung der Grafik notwendig, damit der Leser erkennen kann, womit er es zu tun hat. Dies geschieht meistens durch eine Bildunterschrift, in der zusätzlich eine Abbildungsnummer vergeben wird um im fließenden Text

eindeutig die Grafik referenzieren zu können. Darüber hinaus erlaubt eine solche Referenzierung dem Leser aus dem Text an der richtigen Stelle zur Grafik zu springen, so dass lineares Lesen möglich ist. Eine Grafik, die im Text nicht erwähnt und erklärt wird, kann hingegen im Lesefluss versehentlich übersprungen werden und verliert damit ihren Sinn.

Aber auch schon bei der Auswahl der Grafik sollte überlegt werden, ob das Einfügen an dieser Stelle überhaupt sinnvoll ist. Was bei Texten selbstverständlich ist, gilt auch für Grafiken: Sie sollten übersichtlich erscheinen und eine klare Aussage haben. Der Inhalt muss sich an der Zielgruppe – also dem Leser – orientieren und auf diesen ausgerichtet werden.

3.1.1 Verschiedene Typen von Grafiken

Es gibt viele verschiedene Arten von Grafiken. Bei einem Versuch diese zu klassifizieren, erscheinen folgende Kategorien für sinnvoll:

- Per Hand hergestellte Zeichnung bzw. Bild, welches per Scanner digitalisiert wird.
- Zeichnung auf dem Computer, z. B. mit Paint (Windows), GIMP (GNU).
- Foto (analoges Foto, welches gescannt wird, oder ein mit einer Digitalkamera aufgenommenes Foto, das auf den Computer übertragen wird).
- „Objektorientiert“: Beschreibung einer Grafik durch einzelne Objekte (z. B. mit Adobe Illustrator, Corel Draw).

Mit einer eher technischen Sichtweise beschränkt sich die Klassifizierung auf nur zwei Typen: *Pixelgrafiken* beschreiben in einer Matrix die Farb- und Helligkeitswerte (meist durch Anteile der Grundfarben rot, grün und blau) der einzelnen Bildpunkte (*pixel*; als Abkürzung für engl. *picture element*). Das Skalieren von Pixelgrafiken stellt ein Problem dar, da die Datei, insbesondere beim Vergrößern, nicht die dafür notwendigen Daten beherbergt – die Pixel der neuen Grafik müssen interpoliert werden, was meist zu einem unschönen Resultat führt. Zu diesem Typ gehören die ersten drei der obigen Einteilung.

Bei *Vektorgrafiken* werden hingegen keine Pixel gespeichert, sondern Objekte mit Parametern und Vektoren über deren Position in der Grafik. Zu einer Linie wird beispielsweise

Position, Richtung, Länge, Dicke und Farbe gespeichert. Der Ausgabealgorithmus berechnet dann, an welcher Position ein Pixel gefärbt werden muss um das Objekt auf dem gewählten Ausgabegerät darzustellen. Dadurch kann eine Vektorgrafik ohne Verluste skaliert werden.

3.1.2 Anforderungen an die Textsatz-Software

Um eine reibungslose Nutzung von Grafiken gewährleisten zu können, muss man einige Punkte betrachten, die hierbei Schwierigkeiten – meist technischer Art – bereiten können und diese umgekehrt als Anforderung an die verwendete Textsatz-Software stellen.

Zunächst wird erwartet, dass die eingebundenen Grafiken auf allen Plattformen (d. h. Rechnerarchitekturen und Betriebssystemen) in gleicher Art und Weise dargestellt werden können. Dass dies nicht ohne weiteres der Fall sein kann, lässt die Vielfalt an Dateiformaten für Grafiken erahnen.

Texte inklusive darin eingebundener Grafiken sollen häufig auf verschiedenen Medien ausgegeben werden, die möglicherweise sehr unterschiedliche Auflösungseigenschaften besitzen. Unnötige Arbeit kann vermieden werden, wenn sich die Größe einer Grafik anpassen lässt. Im vorherigen Abschnitt wurde festgestellt, dass Pixelgrafiken dazu ungeeignet sind.

Der in einer Grafik enthaltene Text stellt besondere Ansprüche. Beim Skalieren einer solchen Grafik kann es sinnvoll sein, dass der Text gerade nicht mit skalieren darf. Statt dessen muss er seine Größe beibehalten, damit er gut lesbar bleibt. Ohnehin wird desweiteren erwartet, dass ein Text innerhalb einer Grafik dem Schriftbild des restlichen Dokumentes angepasst ist, d. h. mit den gleichen Eigenschaften gesetzt wird, um ein einheitliches Gesamtbild zu präsentieren.

3.2 Grafiken in LaTeX einbinden

In diesem Abschnitt wird gezeigt, wie bereits vorhandene externe Grafikdateien in LaTeX eingebunden und mit Hilfe von zusätzlichen Paketen positioniert werden. Zu Beginn gibt es jedoch etwas Theorie.

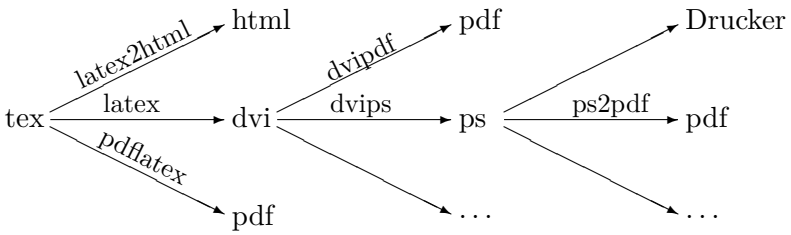


Abbildung 3.1: Das LaTeX-Treibermodell

3.2.1 Technischer Hintergrund

Um Schwierigkeiten beim Einbinden von Grafiken in LaTeX entgegenzuwirken, sollten zumindest ein paar Fakten bekannt sein, wie LaTeX es möglich macht, Grafiken in Dokumente einzubinden.

Das Treibermodell von LaTeX

In LaTeX ist die Verarbeitung einer Eingabedatei in mehrere Schritte, welche von unterschiedlichen Programmteilen durchgeführt werden, aufgeteilt. Zunächst verarbeitet LaTeX die Eingabedatei und erstellt daraus eine geräteunabhängige DVI-Datei (*device independant*), welche bereits den gesetzten Text enthält. Programme, die Treiber genannt werden, wandeln diese nun in systemspezifische Ausgabedateien um (z. B. PostScript zum Druck oder PDF). Diese Dateien können nun direkt zum Ausdrucken oder Anzeigen auf dem Bildschirm genutzt oder durch weitere Hilfsprogramme in andere Dateitypen konvertiert werden. Abbildung 3.1 veranschaulicht dieses Verfahren und nennt einige dafür zu benutzende Programme.

Einbinden von Grafiken

Das direkte Einbinden einer Grafikdatei ist in LaTeX nicht vorgesehen. Statt dessen wird dies über Umwege ermöglicht, die hier kurz vorgestellt werden. In der Praxis abstrahieren Zusatzpakete allerdings von diesen technischen Details.

Das Kommando `\special` erlaubt es Code direkt an den Ausgabetreiber weiterzugeben. Auf diese Weise kann die Fähigkeit der Ausgabetreiber, Grafiken darzustellen, genutzt werden. Die Konsequenz ist, dass der auf diese Weise generierte Code plattformabhängig

Treiber	Ausgabe	Bildformate
dvips	PS (PostScript)	EPS
pdflatex	PDF	PDF, JPEG, PNG

Tabelle 3.1: Unterstützte Dateitypen der Treiber

ist. Insbesondere können so nur solche Grafiken eingebunden werden, die vom Ausgabetreiber verarbeitet werden können (siehe Tabelle 3.1).

LaTeX bietet allerdings die Möglichkeit zusätzliche Schriften einzubinden. Dies kann auf verschiedene Weisen zum Einbinden einer Grafik nützen. Zunächst kann man die komplette Grafik als Ganzes in eine Schriftart konvertieren. Ein Symbol in dieser Schriftart entspricht dann gerade der Grafik. Durch Auswahl der Schriftart und Eingabe des Symbols – der Einfachheit halber ein Buchstabe – wird dann eine Ausgabe der Grafik erzeugt. Diese Methode funktioniert jedoch bei vielen und großen Grafiken nicht sehr gut.

Anstatt die komplette Grafik als Schriftart einzubinden, kann eine Grafik aber auch aus einzelnen Symbolen einer Schriftart zusammengesetzt werden. Dabei werden vorhandene Schriften genutzt, die beispielsweise aus Linien und Pfeilen verschiedener Längen, Dicken und Steigungen bestehen. Aus diesen Grundelementen können dann komplexere Elemente wie Kreise oder Rechtecke zusammengesetzt werden. Dieses Verfahren wird u. a. bei der in Abschnitt 3.3.1 vorgestellen `picture`-Umgebung verwendet.

Eher theoretischer Natur ist das Verfahren eine Schriftart zu nutzen, die aus Punkten verschiedener Helligkeitsabstufungen besteht. Die Grafik wird dann durch Aneinanderreihung der einzelnen Bildpunkte dargestellt.

3.2.2 Einbinden einer Grafikdatei

Eine einfache Variante externe Grafikdateien direkt in LaTeX einzubinden bieten die Pakete `graphics` und `graphicx`. Das `graphicx`-Paket bietet dabei einige zusätzliche Möglichkeiten als das ältere `graphics`-Paket. Beide Pakete hingegen erfüllen die Grundprinzipien des Einbindens von Grafiken, sowie das Rotieren und Skalieren.

Grafikdatei einbinden

Bevor man nun eine Grafikdatei einbindet, sollte man in LaTeX den Quellordner für die Bilddateien angeben. Dies wird durch `\graphicspath{{Verzeichnis/}}` erreicht. *Verzeichnis* gibt hier den relativ zum Ausgangsordner liegenden Ordner an, in dem die Grafikdateien liegen.

Die Grafikdatei wird durch `\includegraphics[option]{dateiname.eps}` eingebunden. Ebenso gibt es hier die Möglichkeit verschiedene Optionen zur Formatierung der Grafik auszuwählen, zum Beispiel

- `draft`, die die eigentliche Grafik in das DVI nicht einbindet, sondern lediglich einen Rahmen in der Größe der Grafik, der den Grafiknamen enthält (geeinget für die Entwurfsphase)
- `scale`, mit der die Grafik mit Werten relativ zur 1 skaliert wird
- `angle`, die die Grafik um eine entsprechende Grad-Zahl dreht

Automatisches Konvertieren von Grafikdateien

Wie schon erwähnt spielt der Ausgabetreiber bei der Wahl der einzubindenden Dateiformate eine entscheidende Rolle. Es gilt also darauf zu achten, welche Dateiformate zum gewünschten Ausgabetreiber passen. LaTeX bietet hier jedoch auch eine Möglichkeit, die Dateiformate automatisch zu konvertieren. Dazu muss zu Beginn des LaTeX-Dokumentes der Befehl `\DeclareGraphicsRule{<Endung>}{<Typ>}{<Groesse>}{<Befehl>}` angegeben werden. *Endung* gibt hier die Endung der Quelldateien an, die konvertiert werden sollen. *Typ* steht für das Zieldateiformat. *Grösse* gibt die gewünschte Grösse des Bildes an. Dies wird über die Angabe einer `BoundingBox` erreicht, wobei dieses Feld alternativ auch leer gelassen werden kann. *Befehl* gibt das Kommando an, welches ausgeführt werden soll um die Grafik in das gewünschte Format zu konvertieren.

Die `figure`-Umgebung

Das Einbinden einer Grafik erfolgt am einfachsten in einer Umgebung, z. B. der `figure`-Umgebung. Auf die einzelnen Gründe und Vorteile der `figure`-Umgebung wird im nächs-

ten Abschnitt eingegangen.

Beim Öffnen der `figure`-Umgebung können verschiedene Optionen für die Positionierung der Grafik angegeben werden, wie zum Beispiel `h` für *here*, damit versucht LaTeX die Grafik an dieser Stelle zu positionieren, `t` für *top* für die Positionierung am Anfang einer Seite und `b` für *botton* für die Positionierung am Ende einer Seite.

Über `\centering` wird die Grafik zentriert. Der Befehl `\caption{Bezeichnung}` erzeugt eine Bildunterschrift mit dem Titel *Bezeichnung*, gleichzeitig wird eine Abbildungsnummer vergeben.

LaTeX bietet die Möglichkeit sogenannte Labels zu erzeugen um im Text des Dokumentes durch einen Befehl Bezug auf eine Grafik nehmen zu können. Entscheidend dabei ist, dass auch Veränderungen der Grafik (zum Beispiel Verschiebung der Grafik in Hinsicht auf die Seitenzahl) automatisch berücksichtigt werden. Dazu wird unter dem Erzeugen der Bildunterschrift der Befehl `\label{Labelname}` hinzugefügt. Somit kann nun im gesamten Dokument durch `\ref{Labelname}` Bezug auf die Grafik und mit `\pageref{Labelname}` auf die Seitennummer nehmen. Der Befehl `\listoffigures` letztlich erstellt am Ende des Dokumentes ein Abbildungsverzeichnis.

```
\usepackage{graphicx}

\begin{figure}[optionen]
  \centering
  \includegraphics{fbi.eps}
  \caption{Fachbereich Informatik}
  \label{fbi-logo}
\end{figure}

Dieser Text bezieht sich auf Abbildung~\ref{fbi-logo} welche auf
Seite~\pageref{fbi-logo} steht.

\listoffigures
```

Positionierung der Grafik

Hier soll nun auf die Positionierung der Grafik eingegangen werden, d. h. welche Auswirkungen hat die Einbindung einer Grafik innerhalb einer `figure`-Umgebung und was passiert, wenn man eine Grafik direkt im Dokumenttext einbindet.

Generell ist der erste Weg der bessere. Bis auf wenige Ausnahmen sollte eine Grafik immer innerhalb einer Umgebung eingebunden werden. Dadurch wird sie in LaTeX als *Gleitobjekt* behandelt. Das bedeutet, LaTeX kann die Positionierung der Grafik variieren und den für sie besten Platz im Dokument bestimmen und dort platzieren. Zwar steht die Grafik dann nicht immer direkt an der vom Benutzer gewünschten Position, jedoch schon in Hinsicht auf die äußere Erscheinung des gesamten Dokumentes. Durch die Deklaration als Gleitobjekt werden zum Beispiel unschöne Zeilenumbrüche und unnötiger Freiraum neben der Grafik verhindert.

3.2.3 Grafiken von Text umfließen lassen

Wie schon im vorigen Abschnitt angesprochen, kann neben einer eingebundenen Grafik unnötiger Freiraum entstehen, der eigentlich von Text genutzt werden könnte. Dazu gibt es in LaTeX die Lösung eine Grafik von Text umfließen zu lassen. Auch hier stehen wieder verschiedene Pakete zur Realisierung zur Verfügung. Im Weiteren werden zwei mögliche Pakete vorgestellt: `picins` und `wrapfig`.

Das Paket picins

`picins` bietet, wie schon erwähnt die Möglichkeit eine beliebige Grafik, zum Beispiel Bilder, Diagramme, aber auch Formeln von Text umfließen zu lassen. Zu Beginn muss natürlich das Paket `picins` eingebunden werden. Der entscheidende Befehl des `picins`-Paketes ist der `\parpic` Befehl. Um eine Grafikdatei, die von Text umflossen werden soll einzubinden, wird der Befehl `\includegraphics{Dateiname} \parpic` übergeben: `\parpic[r]{\includegraphics{Dateiname}}`. Optional kann entschieden werden, wo die Grafik positioniert werden soll, z. B. rechts (`r`) oder links (`l`) und ob die Grafik einen Rahmen bekommen soll (`f`). Im folgenden soll eine mathematische Formel von Text umflossen werden. Zusätzlich soll innerhalb des Rahmens eine Bildunterschrift stehen. Die Bildunterschrift wird durch `\piccaption{Bildunterschrift}` erzeugt. Wichtig dabei

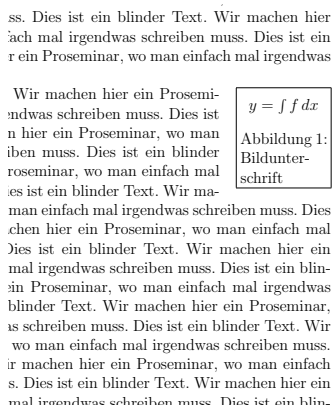


Abbildung 3.2: Picins

ist, dass hier (anders als bei der `figure`-Umgebung) dieser Befehl *vor* `\parpic` bzw. dem Befehl zum Einbinden der Grafikdatei steht! Weiterhin lässt sich die Position der Bildunterschrift durch Befehle wie `\piccaptioninside` (Bildunterschrift innerhalb der Rahmens), `\piccaptionside` (Bildunterschrift neben des Rahmens) festlegen. Abbildung 3.2 zeigt ein Beispiel zu folgendem Quellcode:

```
\usepackage{picins}
...Text...
\piccaptioninside
\piccaption{Bildunterschrift}
\parpic(2cm,1cm)[rf]{
  $y=\int f\,dx$
}
...Text...
```

Das Paket wrapfig

Wie schon bei `picins` gezeigt wurde, ist es möglich Grafiken von Text umfließen zu lassen. Das Paket `wrapfig` erweitert dieses um die Möglichkeit eine Grafik in den Seitenrand zu schieben. Dazu wird beim Beginn einer `wrapfigure`-Umgebung als Parameter die gewünschte Position der Grafik angegeben. Zum Beispiel `L` für das Schieben in den linken bzw. `R` in den rechten Seitenrand. Hierbei ist Groß- bzw. Kleinschreibung entscheidend. Bei Kleinschreibung wird es exakt an der Stelle in den Seitenrand geschoben, an der der

Abbildung 3.3: Wrapfig

Befehl steht. Bei Großschreibung wird es von LaTeX als Gleitobjekt an der besten Stelle positioniert. Der komplette Befehl ist `\begin{wrapfigure}[w]{x}[y]{z}`.

- `w` ist die Anzahl der senkrechten Zeilen, die neben der Grafik begrenzen sollen (optional)
- `x` ist die bereits erwähnte Position der Grafik
- `y` ist die Breite, die die Grafik in den Seitenrand geschoben werden soll (optional)
- `z` ist die Breite der Grafik.

Eine Möglichkeit, wie eine Grafik innerhalb eines Textes in den Seitenrand geschoben wird, zeigt Abbildung 3.3, basierend auf folgendem Code.

```
\usepackage{wrapfig}
 ...Text...
\begin{wrapfigure}{L}[2cm]{5cm}
  \includegraphics{fbi}
\end{wrapfigure}
...Text...
```

3.3 Grafiken mit LaTeX erstellen

In LaTeX können nicht nur externe Grafikdateien eingebunden werden, sondern auch neue Grafiken mit Hilfe verschiedener Pakete direkt aus dem LaTeX-Code generiert werden. Da eine ausführliche Vorstellung aller Pakete den Umfang dieses Kapitels überschreiten würde, wird sich auf eine kurze Vorstellung der `picture`-Umgebung sowie der Pakete `pstricks`, XY-pic, `chess`, und `musixtex` beschränkt. Dem geneigten Leser sei zum Einstieg in tiefergehende Literatur [GRM97] empfohlen.

3.3.1 Die `picture`-Umgebung

Die einzige direkt in LaTeX integrierte Möglichkeit Grafiken zu erstellen bietet die `picture`-Umgebung. Um grafische Objekte darzustellen werden Schriftarten verwendet, die aus elementaren Bildteilen wie Pfeilen und Linien bestehen (siehe Abschnitt 3.2.1). Mit deren Hilfe erzeugt LaTeX simple Bildobjekte wie Rechtecke, Kreise oder Bezier-Kurven. Auch Texte können eingebunden werden, die wie der restliche Text im Dokument gesetzt werden – allerdings ohne Zeilenumbruch. Da kein treiberabhängiger Code verwendet wird, ist diese Methode plattformunabhängig.

Positions- und Längenangaben werden in der gesamten `picture`-Umgebung über eine benutzerdefinierte Einheit geregelt. Dessen Größe wird in dem Längenregister `\unitlength` festgelegt. Innerhalb der Umbegung können nur ganzzahlige Vielfache dieser Einheit angegeben werden, daher sollte die Einheit klein genug gewählt werden.

Als Parameter wird beim Öffnen der `picture`-Umgebung die Größe der Grafik als Tupel angegeben. Optional kann ein weiteres Tupel den Nullpunkt zur weiteren Referenzierung der Objekte definiert werden. Diese werden dann mit `\put(x,y){Objekt}` oder `\multiput(x,y)(dx,dy){n}{Objekt}` positioniert – mit letzterem lässt sich ein Objekt `n`-mal, jeweils mit einer Verschiebung um den Vektor (`dx`, `dy`), darstellen.

Nur wenige Objekttypen lassen sich innerhalb dieser Umgebung ohne weitere Meta-Pakete darstellen. Beispielsweise kann mit `\line(dx,dy){länge}` eine Linie gezeichnet werden, wobei das Tupel (`dx`, `dy`) die Steigung der Linie definiert. `\vector` besitzt die gleichen Parameter mit selber Semantik, zeichnet allerdings eine Linie mit Pfeilspitze. Es gibt noch weitere Befehle, die sich ähnlich intuitiv benutzen lassen. Das folgende Beispiel demonstriert (siehe Abbildung 3.4) die Anwendung einiger.

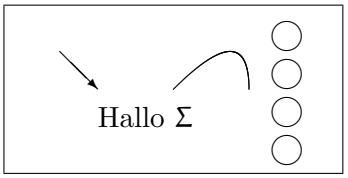


Abbildung 3.4: Beispiel zur `picture`-Umgebung

```
\setlength{\unitlength}{0.1cm}
\fbox{\begin{picture}(40,20)(0,0)
  \put(10,5){Hallo $\Sigma$}
  \put(5,15){\vector(1,-1){5}}
  \put(20,10){\qbezier(0,0)(10,10)(10,0)}
  \multiput(35,17)(0, -5){4}{\circle{4}}
\end{picture}}
```

3.3.2 PSTricks

PostScript wurde 1984 von Adobe als eine plattformunabhängige Seitenbeschreibungssprache entwickelt. Heute ist es quasi die Standardsprache für professionellen Druck. PostScript ist sehr mächtig: statt „nur“ eine Beschreibungssprache zu sein, verfügt PostScript über das Potenzial einer vollständigen Programmiersprache. Das Paket `pstricks` stellt eine benutzerfreundliche Schnittstelle von LaTeX zu PostScript bereit. Selbstverständlich können dessen Fähigkeiten nur von Ausgabetreibern genutzt werden, die PostScript verstehen, wie `dvips` oder `xdvi` – letzteres verfügt über diese jedoch nur in eingeschränktem Maße.

Zusätzlich zu den grundlegenden Befehlen dieses Paketes existieren höhere Pakete in der PSTricks-Distribution, welche abstraktere Fähigkeiten besitzen. Dazu gehören `pst-3d` zum Erstellen von dreidimensionalen Objekten, `pst-node` zum Erstellen von (mathematischen) Graphen, `pst-plot` zum Darstellen von Funktionsgraphen und Diagrammen, `pst-tree` zum Erstellen von Baumstrukturen, `pstcol` für farbige Darstellungen und noch viele mehr. Wird eines dieser Pakete in das LaTeX-Dokument eingebunden, so muss nicht noch zusätzlich `pstricks` geladen werden – dies geschieht automatisch.

Dass `pstricks` sehr umfangreich ist, zeigt schon das etwa 150 Seiten umfassende Handbuch [Zan]. Dieser Abschnitt ist daher lediglich als Motivation zu verstehen, tiefergehen-

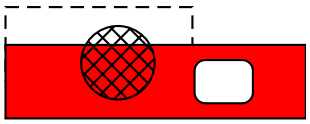


Abbildung 3.5: Objekte mit PSTricks

de Literatur zu konsumieren.

Geometrische Objekte zeichnen

Ähnlich wie in der `picture`-Umgebung lassen sich auch mit PSTricks simple Objekte einfach darstellen. Die Syntax ist selbstverständlich eine andere. So werden bei PSTricks die Makros innerhalb einer `pspicture`-Umgebung geschrieben, die als Parameter die Größe der Grafik entgegennimmt. Als Größeneinheit wird standardmäßig 1 cm verwendet, dies kann jedoch über die Variable `unit` geändert werden. Dazu wird das Makro `\psset` verwendet, etwa `\psset{unit=1mm}`. Viele PSTricks-Makros bieten über einen optionalen Parameter die Möglichkeit mittels Schlüssel-Wert-Paare in der Form `option=wert` weitere Attribute, beispielsweise die Füllungsart eines Rechtecks, den Linienstil eines Kreises und die zu verwendenen Farben, zu spezifizieren. Abbildung 3.5 zeigt das Resultat des folgenden Beispielcodes.

```
\usepackage{pstcol}
\definecolor{rot}{rgb}{1,0,0}
\begin{pspicture}(4.5,2)
  \psframe[linestyle=dashed](0.5,0.5)(3,1.5)
  \psframe[fillstyle=solid,fillcolor=rot](.5,0)(4.5,1)
  \psframe[fillstyle=solid,fillcolor=white,
           framearc=0.5](3,0.2)(3.8,0.8)
  \pscircle[fillstyle=crosshatch](2,0.75){0.5}
\end{pspicture}
```

Objekte maskieren

PSTricks gestattet anhand der Kontur eines Objektes eine (Teil-)Grafik derart zu beschneiden, wie Abbildung 3.6 zeigt. Dazu wird dem Makro `\psclip` im Parameter ein

Abbildung 3.6: Maskieren mit PSTricks

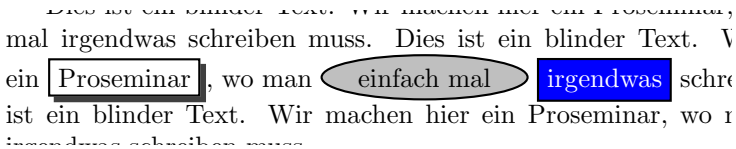


Abbildung 3.7: Texte auszeichnen mit PSTricks

Objekt übergeben – in diesem Fall eine Raute – anhand dem alles bis zu `\endpsclip` liegende maskiert wird.

```
\usepackage{pstcol}
...
\begin{pspicture}(-1,0)(2,2)
  \psclip{\psdiamond(1.5,0.2)(2.0,0.5)}
  \Huge Hallo Welt
  \endpsclip
\end{pspicture}
```

Text auszeichnen

Einige Fähigkeiten lassen sich sogar außerhalb einer `pspicture`-Umgebung nutzen. So stellt PSTricks Makros bereit, mit denen sich Passagen im fließenden Text auf verschiedene Weisen markieren lassen: `\psovalbox` zeichnet ein Oval um den Text bzw. das Objekt im Parameter, `\psframebox` ein Rechteck. Mit `\color` kann man einen Text sogar einfärben, wie in Abbildung 3.7 gezeigt wird.

```
\usepackage{pstcol}
...
... ein \psshadowbox{Proseminar}, wo man
\psovalbox[fillcolor=lightgray,
           fillstyle=solid]{einfach mal}
```

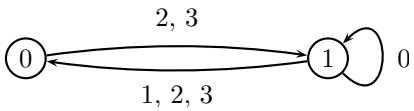


Abbildung 3.8: Endlicher Automat mit PSTricks

```
\psframebox[fillcolor=blue,fillstyle=solid]
{\color{white}irgendwas} schreiben ...
```

Knoten verbinden

Ein höheres Paket, welches sich beispielsweise zum Zeichnen von endlichen Automaten aus der formalen Informatik eignet, ist `pst-node`. Es stellt u. a. verschiedene Makros zum Positionieren von Knoten (z. B. `\cnodeput(x,y){Name}{Beschriftung}`), zum Verbinden der Knoten mit Kanten (z. B. `\ncarc{Knoten1}{Knoten2}`) und zum Beschriften der Kanten (z. B. `\naput` für oberhalb bzw. `\nbput` für unterhalb der Kante) zur Verfügung. Abbildung 3.8 zeigt das Ergebnis des folgenden Beispiels.

```
\usepackage{pst-node}
...
\begin{pspicture}(-1.5,-1)(7.5,4.5)
  \psset{arrows=->,shortput=nab}
  \cnodeput(0,0){q0}{0}
  \cnodeput(4,0){q1}{1}
  \ncarc{q0}{q1}\naput{2, 3}
  \ncarc{q1}{q0}\naput{1, 2, 3}
  \nccurve[ncurv=5,angleA=315,angleB=45]{q1}{q1}\nbput{0}
\end{pspicture}
```

3.3.3 XY-pic

Das Paket XY-pic unterstützt die Erstellung von Diagrammen und Graphen mit Mitteln von METAFONT und LaTeX. Es ist ein sehr mächtiges Werkzeug mit dem vielfältigte Arten von Graphen mit optionalen Paketbestandteilen darstellbar sind. Daher werden nur einige Grundelemente und wenige Beispiele erklärt.

Das Paket wird im Kopf des Dokuments eingebunden (die Option *all* sorgt dafür, dass alle Komponenten des Pakets unterstützt werden): `\usepackage[all]{xy}`. Eine häufig verwendete Form sind Matrizen mit Pfeilen zwischen den Einträgen (`\xymatrix{...}`). Die Spalten werden durch `&`, die Zeilen durch `\\` getrennt. Pfeile werden an dem Eintrag angegeben an dem sie starten und haben die folgende Form:`\ar@{Art}[Richtung]`. *Art* definiert das Aussehen des Pfeils, z. B. `->` für einen einfachen Pfeil, `<=>` für einen Doppelpfeil mit zwei Spitzen, `->` für einen gestrichelten Pfeil oder `-` für eine einfach Linie etc. Wird der Teil *@Stil* weggelassen, dann wird ein einfacher Pfeil erzeugt. Die *Richtung* ist eine Sequenz der Buchstaben `r` (für right), `l` (für left), `u` (für up) und `d` (für down) und gibt den relativen Endpunkt des Pfeils an.

Eine Beschriftung des Pfeils wird erzeugt, indem man einen (oder mehrere) der folgenden Befehle an das Kommando `\ar` anhängt:

- `^{Beschriftung}` für eine Beschriftung oberhalb des Pfeils
- `_{Beschriftung}` für eine Beschriftung unterhalb des Pfeils
- `|{Beschriftung}` für eine Beschriftung in der Mitte des Pfeils

Im folgenden Beispiel soll nun ein Diagramm mit XY-pic gezeichnet werden. Das Ergebnis ist in Abbildung 3.9 zu sehen.

Zunächst wird das Package XY-pic zu Beginn des Dokumentes eingebunden. Durch den Befehl `\xymatrix{...}` wird eine Matrixstruktur geöffnet. Der erste Eintrag in der ersten Zeile und ersten Spalte ist ein *A*. Durch `\ar[r]^a` wird ein Pfeil vom jetzigen Eintrag nach rechts gezechnet. Durch das `&` wird in die nächste Spalte gesprungen, wo ein *B* mit einem beschrifteten Pfeil selbstgewählter Art nach unten rechts gezeichnet wird (`B \ar@{~>}[dr]^b`). Um einen leeren Eintrag zu bekommen wird nach einem `&` einfach in die nächste Spalte oder in die nächste Zeile(durch `\\`) gesprungen, so wie es im Beispiel gemacht wurde. Dort wird wieder der erste Eintrag fokussiert. Es können natürlich auch mehrere Pfeile von einem Eintrag aus starten, indem mehrere Befehle für Pfeile hintereinander gefügt werden. Hier wird nun ein *C* mit bestimmter Pfeilart nach oben rechts und ein weiterer Pfeil bestimmter Art mit Beschriftung x oberhalb und y unterhalb des Pfeils zwei Einträge nach rechts(durch die Richtungssequenz *rr*) gezeichnet (`C \ar@{.>}[ur] \ar@{=>}[rr]^x_y`). Letztlich wird widerrum ein Eintrag

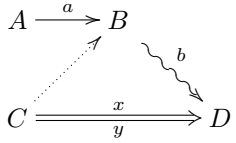


Abbildung 3.9: Diagramm mit XY-pic

leer gelassen und der letzte Eintrag mit *D* gefüllt. Das gesamte Beispiel ist im folgenden Code noch einmal zusammenfasst:

```
\usepackage[frame,curve,arrow,matrix]{xy}
\begin{document}
\xymatrix{A \ar[r]^a & B \ar@{~>}[dr]^b &\\
C \ar@{.>}[ur] \ar@{=>}[rr]^x_y && D}
\end{document}
```

XY-pic eignet sich auch sehr gut für Zustandsdiagramme, wie die Abbildung 3.10 zu folgendem Quellcode zeigt. Anders als im vorigen Beispiel wurde hier vorab bestimmt, wie die Einträge in der Matrix aussehen sollen, d. h. wenn nicht explizit mitgeteilt wird, wie die Einträge aussehen sollen, wird immer das Standardformat genommen. Ein Standardformat für die Einträge wird duch `\entrymodifiers={++[o][F-]}` gesetzt. Der erste Parameter gibt die Art des Eintrages an, wie hier z. B. o für einen Kreis. Der zweite Parameter *F-* steht für einfach eingerahmt. Weitere Möglichkeiten wären *F=* für doppelt eingerahmt, *F.* für gepunkteten Rahmen, *F–* für getrichelten Rahmen, etc.

```
\usepackage[frame,curve,arrow,matrix]{xy}
...
\entrymodifiers={++[o][F-]}
\xymatrix{*\txt{Start} \ar[r] &
1 \ar[r]^a &
2 \ar[r]_b \ar@(r,u)[]_c &
3 \ar[r]^c \ar ‘d_l[ll] ‘_u[ll] ^x [ll]&
*++[o][F=]{4} }
```

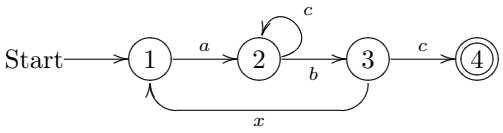


Abbildung 3.10: Zustandsgraph mit XY-pic

3.3.4 Ein Ausblick: Zwei weitere Pakete

Zum Abschluss wird kurz gezeigt, dass in LaTeX auch andere Themen als Diagramme, Graphen oder externe Grafiken dargestellt werden können.

Chess

Ein Paradebeispiel für ein Paket, welches Schriftarten nutzt um eine Grafik darzustellen, ist `chess`. Es bringt eine Schriftart mit, die u. a. alle Figuren des Schachspieles in beiden Farben schwarz und weiß beinhaltet. So lassen sich sehr einfach Schachsituationen darstellen. Um das in Abbildung 3.11 gezeigte Brett zu erzeugen nutzt `chess` eine Matrix innerhalb der LaTeX-Matheumgebung. Nach Auswahl der Schach-Schriftart (hier `chess20`) wird über das Makro `\board` das Feld spezifiziert. Die Buchstaben stehen für die einzelnen Figuren (Kleinbuchstaben für schwarz, Großbuchstaben für weiß), ein Leerzeichen für ein weißes Feld und ein Sternchen für ein schwarzes Feld. Die Ausgabe des Brettes erfolgt durch `\showboard` innerhalb einer Matheumgebung.

```
\usepackage{chess}
\font\Chess=chess20
\board{rnbqkbnr}
      {pppppppp}
      { * * * *}
      {* * * * }
      { * * * *}
      {* * * * }
      {PPPPPPPP}
      {RNBQKBNR}
$$\showboard$$
```

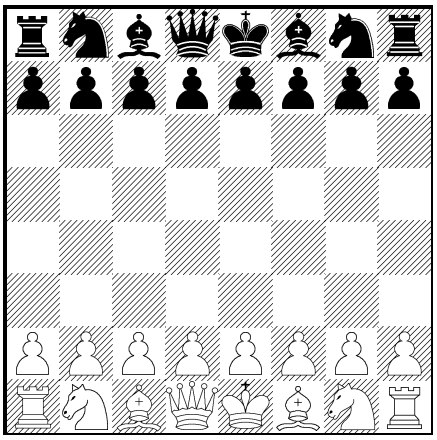


Abbildung 3.11: Der Beginn eines Schachspiels

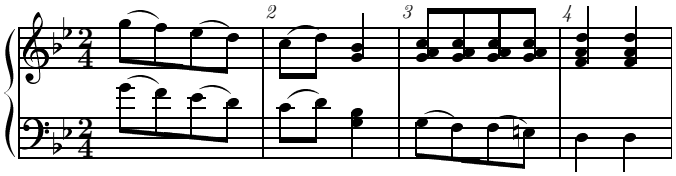


Abbildung 3.12: Darstellung von Noten

MusiXTeX

Das Paket MusiXTeX dient zur Darstellung von Noten und Notenlinien. Es können mehrzeilige Notenlinien, beispielsweise für die Darstellung von Noten für Violin- und Bassschlüssel, erzeugt werden. Das Paket lässt nahezu keine Wünsche offen. Leider ist es daher sehr komplex, was die Noteneingabe nicht gerade einfach gestaltet. Es gibt jedoch Zusatzsoftware, welche von den MusiXTeX-Makros abstrahiert indem eine eigene, einfachere Syntax verwendet wird. Die Software konvertiert dann eine Eingabedatei in LaTeX-Code. Das folgende Beispiel ist aus [GRM97] übernommen, Abbildung 3.12 zeigt das Ergebnis.

```
\usepackage{musixtex}
\begin{music}
\instrumentnumber{1}
\setstaffs{1}{2}
\generalmeter{\meterfrac24}
```

```
\setclef{1}{\bass}
\setclef{2}{\treble}
\generalsignature{-2}
\startextract
\Notes
  \isluru0g\ibl0e{-2}\qb0{g}\tslur0f\qb0{f}%
  \isluru0e\qb0{e}\tslur0{d}\tbl0\qb0{d}%
 |\isluru0n\ibl0m{-2}\qb0{n}\tslur0m\qb0{m}%
  \isluru0l\qb0{l}\tbl0\tslur0k\qb0{k}%
\enotes\bar
\Notes
  \isluru0c\ibl0c0\qb0{c}\tslur0d\tbl0\qb0{d}%
 |\isluru0j\ibl0j0\qb0{j}\tslur0k\tbl0\qb0{k}%
\enotes
\NOtes \zq{N}\ql{b}|\zq{g}\qu{i}\enotes\bar
\Notes
  \isluru0N\ibl0N{-2}\qb0{N}\tslur0M\qb0{M}%
  \isluru0M\qb0{M}\tslur0L\tbl0\qb0{=L}%
 |\ibu0j0\zqb0{g}\rq{h}\qb0{j}\zqb0{g}\rq{h}\qb0{j}%
  \zqb0{g}\rq{h}\qb0{j}\tbu0\zqb0{g}\rq{h}\qb0{j}%
\enotes\bar
\NOtes \ql{K}\ql{K}|\zq{f}\zq{h}\qu{k}\zq{f}\zq{h}\qu{k}\enotes
\endextract
\end{music}
```

Kapitel 4

Index und Bibliographie

Moritz Gerner-Beuerle, Mario Maiworm, Nicolas Paul

In diesem Kapitel werden die wichtigsten Möglichkeiten vorgestellt, mit LaTeX Indices und Bibliographien zu erstellen.

4.1 Bibliographieren

Moritz Gerner-Beuerle

> I respect a man who can recognize a quotation. It's dying art.
>
> (David Lodge, *Small world*)

4.2 Bibliographie - Definition, Aufgaben und Schwierigkeiten

4.2.1 Einführung und Definition

Unter Bibliographie versteht man grundsätzlich im wissenschaftlichen Sprachgebrauch die Handlung der Erfassung der einer wissenschaftlichen Arbeit zugrunde liegenden Literatur. Dies untergliedert sich in die Sichtung der zum Thema vorhandenen Literatur, sowie in die Erfassung der bibliographischen Angaben der verwendeten Literatur. Ebenso sind mit Bibliographie die Lehre von dieser Tätigkeit als auch das Ergebnis, also ein

Literaturverzeichnis, gemeint. Das folgende Kapitel legt den Schwerpunkt auf die Betrachtung des technischen Vorgangs des Bibliographierens und die Aufnahme der bibliographischen Angaben in einer wissenschaftlichen Arbeit, die mit LaTeXverfasst wurde.

4.2.2 Zweck des Bibliographierens

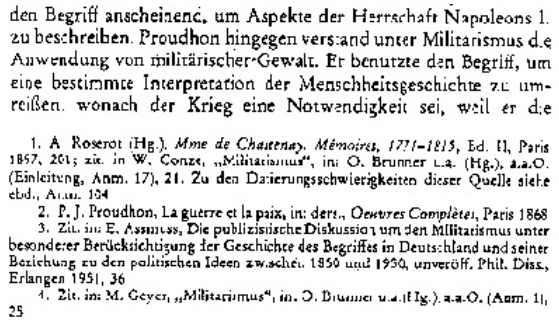
Bibliographieren ist eine der wichtigsten Aufgaben für den Autor einer wissenschaftlichen Arbeit oder eines wissenschaftlichen Berichts. Schon in der Antike wurde kein Text als *creatio ex nihilo* (aus dem Nichts heraus) [vdB99, S. 13] begriffen. Zum einen dienen bibliographische Angaben dem Betrachter einer Arbeit dazu, die Ergebnisse derselben zu prüfen und nachzuvollziehen. Durch die Auswahl, die der Autor an Sekundärliteratur und Quellen getroffen hat, lassen sich Rückschlüsse auf den Standpunkt und das Arbeitsumfeld des Autors ziehen. Unter Umständen lässt sich über das Literaturverzeichnis auch einschätzen, ob die vorliegende Arbeit den Ansprüchen des Lesenden genügen. Der Forschungsstand, auf dem die Arbeit aufbaut, ist, falls nicht gesondert beschrieben, dem Literaturverzeichnis am schnellsten zu entnehmen. Zum anderen trennt eine genaue Bibliographie die Forschungsleistungen des Autors von denen anderer Wissenschaftler ab, auf denen Sie aufbauen. Hier dient die Bibliographie also dazu, die eigenen Leistungen nicht mit denen anderer Wissenschaftler zu vermengen. Ein exaktes Literaturverzeichnis und die Kennzeichnung der Textpassagen, deren Gedanken anderen Arbeiten entstammen, erhöht die wissenschaftliche Glaubwürdigkeit einer Arbeit und damit auch die ihres Autoren. Zusätzlich kann es auch den Wert für einen späteren Konsumenten der Arbeit erheblich steigern, wenn er auf andere Arbeiten aufmerksam gemacht wird. Bibliographie dient dazu, eine Art Netzwerk zwischen verschiedenen wissenschaftlichen Arbeiten zu einem Themenkomplex aufzubauen, in dem sich der Wissenschaftler orientieren kann. In der Literaturwissenschaft wird dies als „Intertextualität“oder auch „Transtextualität“bezeichnet.[vdB99]

4.2.3 Interdisziplinäre Unterschiede

Beim Bibliographieren gibt es zwischen einzelnen wissenschaftlichen Disziplinen Unterschiede in der Auffasung über die Form und Struktur der Bibliographie einer wissenschaftlichen Arbeit. So wird in der Informatik gelehrt, eine Literaturangabe im Text mittels Querverweis zum im Anhang befindlichen Literaturverzeichnis zu machen. In der

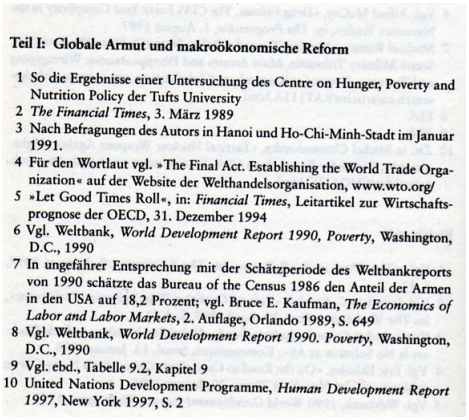
Geschichtswissenschaft hingegen wird in einer Fussnote die Literatur mit allen Angaben aufgeführt, zusätzlich zu einem Literaturverzeichnis im Anhang. Das Literaturverzeichnis gehört bei wissenschaftlichen Arbeiten im deutschsprachigen Raum in den Anhang, das ist interdisziplinärer Konsens. Nur bei Sammelbänden kommt es vor, das es nach jedem Aufsatz ein eigenes Literaturverzeichnis gibt.

Auch in der Form der einzelnen Literaturangaben gibt es Uneinigkeit. So werden in der Psychologie in der Form `Autor(en). Jahr. Titel. Erscheinungsort. Verleger.`, in der Geschichtswissenschaft in der Form `Autor(en), Titel, Verleger, Erscheinungsort, Jahr.` Literaturangaben aufgenommen. Die folgenden Grafiken machen die Unterschiede deutlich.



den Begriff anscheinend, um Aspekte der Herrschaft Napoleons 1. zu beschreiben. Proudhon hingegen verstand unter Militarismus die Anwendung von militärischer Gewalt. Er benutzte den Begriff, um eine bestimmte Interpretation der Menschheitsgeschichte zu umreißen, wonach der Krieg eine Notwendigkeit sei, weil er die

1. A. Roserot (Hg.), *Mme de Chastenay. Mémoires, 1771–1815*, Bd. II, Paris 1897, 261; zit. in W. Conze, „Militarismus", in: O. Brunner u.a. (Hg.), a.a.O. (Einleitung, Anm. 17), 21. Zu den Datierungsschwierigkeiten dieser Quelle siehe ebd., Anm. 104
2. P. J. Proudhon, La guerre et la paix, in: ders., *Oeuvres Complètes*, Paris 1868
3. Zit. in: E. Assmuss, Die publizistische Diskussion um den Militarismus unter besonderer Berücksichtigung der Geschichte des Begriffes in Deutschland und seiner Beziehung zu den politischen Ideen zwischen 1850 und 1950, unveröff. Phil. Diss., Erlangen 1951, 36
4. Zit. in: M. Geyer, „Militarismus", in: O. Brunner u.a. (Hg.), a.a.O. (Anm. 1), 25



Teil I: Globale Armut und makroökonomische Reform

1 So die Ergebnisse einer Untersuchung des Centre on Hunger, Poverty and Nutrition Policy der Tufts University
2 *The Financial Times*, 3. März 1989
3 Nach Befragungen des Autors in Hanoi und Ho-Chi-Minh-Stadt im Januar 1991.
4 Für den Wortlaut vgl. »The Final Act. Establishing the World Trade Organization« auf der Website der Welthandelsorganisation, www.wto.org/
5 »Let Good Times Roll«, in: *Financial Times*, Leitartikel zur Wirtschaftsprognose der OECD, 31. Dezember 1994
6 Vgl. Weltbank, *World Development Report 1990, Poverty*, Washington, D.C., 1990
7 In ungefährer Entsprechung mit der Schätzperiode des Weltbankreports von 1990 schätzte das Bureau of the Census 1986 den Anteil der Armen in den USA auf 18,2 Prozent; vgl. Bruce E. Kaufman, *The Economics of Labor and Labor Markets*, 2. Auflage, Orlando 1989, S. 649
8 Vgl. Weltbank, *World Development Report 1990. Poverty*, Washington, D.C., 1990
9 Vgl. ebd., Tabelle 9.2, Kapitel 9
10 United Nations Development Programme, *Human Development Report 1997*, New York 1997, S. 2

4.3 Technische Umsetzung mit LaTeX

4.3.1 Literaturangaben aufnehmen

Die bibliographischen Angaben eines Buches befinden sich in der Regel auf dem Titelblatt, welches auf den ersten drei Seiten eines Buches zu finden ist. Entscheidend ist in jedem Fall das Titelblatt und nicht die Angaben auf dem Einband. Zu den bibliographisch relevanten Angaben zählen vor allem der Name des Autors, wobei im Literaturverzeichnis akademische Titel entfallen, der Titel, Erscheinungsjahr und der Erscheinungsort. Hier ist darauf zu achten, dass man den Ort der Niederschrift des Buches nicht mit dem Ort des Drucks verwechselt. Ebenfalls wichtig ist die Auflage des Werkes, um den Forschungsstand nachvollziehen zu können.

Mit LaTeXbesteht die Möglichkeit, Literaturangaben direkt in das Dokument einzubinden. Hierzu wird am Ende des Quelltext-Dokumentes eine \thebibliography - Umgebung angelegt, in der die Literaturangaben in einer festgelegten Syntax notiert werden. Im Text wird auf die Literatur mit dem \cite - Tag und einem in der \thebibliography - Umgebung angelegten oder generierten Schlüssel verwiesen.

4.3.2 `thebibliography` und `cite`

Im Folgenden wird das Erstellen eines Literaturverzeichnisses am Beispiel zweier Bücher dargestellt. Hierfür werden die Literaturangaben zuerst in der \thebibliography - Umgebung aufgenommen. Diese Umgebung kann natürlich auch in einer externen Datei stehen, die dann mit \input eingebunden wird.

```
\begin{thebibliography}{9999}
\bibitem{Dick01} Dickreiter, Michael:
\emph{Musikinstrumente}, Baerenreiter Verlag, Berlin
\textsuperscript{6}2001.

\bibitem[Goldt98]{Gold98} Goldt, M.:
\emph{Mindboggling - Evening Post}, Hamburg 1998.
\end{thebibliography}
```

Die Syntax ist also wie folgt vorgegeben:

```
\begin{thebibliography}{maximale Key-Stellen}
\bibitem[optionaler Text]{Key}
Literaturangaben mit Formatierung (s. Beispiel)
\end{thebibliography}
```

Für jeden Eintrag ist ein \bibitem zu verwenden, wobei man auf die Einmaligkeit jedes Schlüssels zu achten hat. Der optionale Text nach \bibitem erscheint im Literaturverzeichnis und an der zitierten Stelle im Text als Indexierung vor dem eigentlichen Eintrag. Wird diese Option nicht mit angegeben, werden die Einträge nach der Reihenfolge im Quelltext durchnummeriert.

Die Referenz im Text wird mit \cite{key} hergestellt.

.

```
Ich gebe zu, dass es sich hier nicht um Hopi - Weisheiten handelt. \cite{Gold98}
```

Auch hier ist ein zusätzliches Argument möglich, das in eckigen Klammern direkt hinter \cite gesetzt wird:

```
Ich gebe zu, da"s es sich hier nicht um Hopi - Weisheiten handelt.
\cite[S. 48]{Gold98}
```

Ein Literaturverzeichnis und eine Textstelle mit den beiden oben genutzten Büchern sähen also folgendermaßen aus:

Ich gebe zu, dass es sich hier nicht um Hopi - Weisheiten handelt. [Goldt, S. 48]

Literaturverzeichnis

[1]Dickreiter, Michael: *Musikinstrumente*, Baerenreiter Verlag, Berlin ^{6}2001.

[Goldt]Goldt, Max: *Mindboggling - Evening Post*, Hamburg 1998.

```
nocite
```

Im nächsten Kapitel geht es um BibTEX, das es ermöglicht, eine externe Literaturdatenbank aufzubauen. Mit \nocite{ } kann man einen Eintrag aus einer BibTEX-Datei in das Literaturverzeichnis aufnehmen. Die Zuordnung erfolgt über den Schlüssel des BibTEX-Eintrages.

4.4 BibTEX

Mario Maiworm

In diesem Abschnitt wird das Programmpaket BibTEX vorgestellt. Außerdem wird auf einige Hilfsmittel und Tools eingegangen, welche die Arbeit mit Bibliographien erleichtern.

4.4.1 Begrenzungen der `thebibliography`-Umgebung

Im vorherigen Kapitel wurde die `\thebibliography{}`-Umgebung beschrieben, womit innerhalb eines LaTeX-Dokuments Literaturverzeichnisse erstellt werden können. Sobald man regelmäßig Bibliographien erstellt oder diese einen gewissen Umfang erreichen, werden einige Nachteile der `\thebibliography{}`-Umgebung deutlich:

- Jeder Eintrag im Literaturverzeichnis erscheint im kompilierten Dokument so, wie er im Quelldokument eingegeben wurde. Dadurch ist der Autor selbst für die einheitliche Formatierung des Literaturverzeichnisses verantwortlich. Dies ist arbeitsintensiv und fehleranfällig, zumal es der „LaTeX-Philosophie“ widerspricht, Gliederung und Inhalt von der Formatierung zu trennen.
- Für jedes neu erstellte Dokument muss im Quelltext eigens ein Literaturverzeichnis programmiert werden. Verwendet der Autor dieselben Literaturstellen in mehreren Dokumenten, fällt diese Arbeit mehrfach an (Eine bestimmte Literaturstelle muss in mehreren Dokumenten im Literaturverzeichnis programmiert werden.).
- Sind mehrere Autoren an der Erstellung eines Dokuments beteiligt („collaborative writing“), so stellt sich dasselbe Problem: Prinzipiell ist jeder Autor für sein eigenes Literaturverzeichnis zuständig, sodass mitunter dieselben Literaturangaben mehrmals programmiert werden.

Zur Lösung dieser Probleme ist ein System wünschenswert, womit vom jeweiligen Quelldokument unabhängige „Literaturdateien“ erstellt werden können, die eine Sammlung der zu zitierenden Literaturstellen enthalten. Diese Dateien sollen dann in Quelldokumente eingebunden werden, ohne dass jeweils ein eigenes Literaturverzeichnis vom Autor erstellt werden muss. Dieses Modell kann mit dem Programm BibTeX verwirklicht werden.

4.4.2 BibTeX

BibTeX von Oren Patashnik ist ein Zusatzprogramm zum Erstellen und Verwalten von Bibliographien in LaTeX-Dokumenten. Literaturdaten können hierbei in entsprechende Literaturdateien mit der Dateierweiterung *.bib* eingegeben werden. Auf diese kann innerhalb eines LaTeX-Quelldokumentes verwiesen werden, sodass nicht für jedes Dokument

vom Autor ein eigenes Literaturverzeichnis programmiert werden muss, sondern die *.bib*-Dateien beliebig eingebunden werden können. Unabhängig davon wird das Format der Bibliographien durch Style-Files (**.bst*) vorgegeben. Da hier Vorlagen benutzt werden können, überlässt der Autor im Regelfall LaTeX die Formatierung der Bibliographie.

Erstellen von Bibliographien mit BibTeX

Um mit BibTeX eine Bibliographie in ein LaTeX-Dokument einzubinden, wird der Befehl `\bibliography{}` verwendet. Er steht an der Stelle im Quellcode, wo später das Literaturverzeichnis erscheinen soll. Als obligatorischer Parameter werden die Namen einer oder mehrerer Literaturdateien (**.bib*) übergeben (ohne Dateierweiterung). In den **.bib*-Dateien wird jeder Literaturstelle ein Schlüssel zugewiesen. Referenzen auf eine bestimmte Literaturstelle werden im Quelltext dann mit `\cite{`*Schlüssel*`}` gesetzt. An dieser Stelle wird im kompilierten Dokument ein Verweis auf die jeweilige Literaturstelle erscheinen. Mit `\nocite{`*Schlüssel*`}`wird eine Literaturstelle im Literaturverzeichnis aufgefürt, ohne dass ein Verweis im Dokumenttext darüber erscheint. Der Befehl `\nocite{*}` sorgt dafür, dass alle Literaturangaben aus den angegebenen **.bib*-Dateien im Literaturverzeichnis erscheinen, ohne dass auf sie vorher verwiesen wurde.
Der Befehl `\bibliographystyle{}`übergibt als obligatorischen Parameter die Style-Datei, welche die Formatierung der Bibliographie festlegt.

Im Beispiel (Abbildung 4.1) werden die Literaturdateien *mixedliterature.bib* und *newliterature.bib* eingebunden. Durch `\nocite{*}` werden alle in den beiden Dateien enthaltenen Literaturangaben in das Literaturverzeichnis eingebunden. Die Style-Datei *plain.bst* wird als Formatvorlage für die Bibliographie festgelegt. Durch den Befehl `\cite{2}` wird im kompilierten Text nach „before“ ein Verweis auf die Literaturstelle gesetzt, die in einer der beiden Literaturdateien mit dem Schlüssel *2* spezifiziert ist. Bei der Formatvorlage *plain.bst* sind diese Verweise Zahlen, die Einträge im Literaturverzeichnis sind nach Nachnamen der Autoren alphabetisch geordnet.
Durch einen LaTeX-Aufruf werden die zum Referenzieren erforderlichen Informationen in die Hilfsdatei (**.aux*) geschrieben. Nun muss BibTeX aufgerufen werden. In den meisten LaTeX-Editoren ist dies als Ausführungsoption enthalten, auf Kommandoebene wird der BibTeX-Durchlauf mit *bibtex <datei>* angestossen, wobei *datei* der Name der **.aux*-Datei ist. Das von BibTeX erzeugte Literaturverzeichnis (**.bbl*) wird durch einen wei-

```
\documentclass{article}
.
.
.
\begin{document}
As has been argued before\cite{2}, excessive use of the
\LaTeX -environment bears the risk of clearly formatted
literature sections.
\nocite{*}
\bibliography{mixedliterature,newliterature}
\bibliographystyle{plain}
\end{document}
```

Abbildung 4.1: Beispiel

teren LaTeX-Lauf in das Dokument eingebunden. Ein dritter LaTeX-Lauf ist notwendig, um alle Querverweise herzustellen. Das Dokument liegt danach mit Literaturverzeichnis vor.

4.4.3 Datenbankdateien (*.bib)

Wie bereits erwähnt, müssen die Informationen zu den Literaturstellen in einer **.bib*-Datei vorliegen. Diese „Datenbankdateien“ enthalten für jede Literaturstelle einen Eintrag, wobei diese Einträge nach Art der Literaturangabe unterschieden werden. Dies ist notwendig, weil die Literaturangaben in Feldern gemacht werden und z. B. für einen wissenschaftlichen Artikel andere Felder benötigt werden als für ein Buch. Das Feld *journal* ist für Bücher sinnlos, das Feld *editor* ebenso für Artikel. Aus diesem Grund werden alle vorhandenen Felder für jeden Eintragstyp (Buch, Artikel, ...) in die Kategorien *required*, *optional* und *ignored* unterteilt, wobei Informationen zu Feldern der Kategorie *required* obligatorisch sind, *optional* bedeutet, dass die Felder angegeben werden können, und Informationen in den Feldern der Kategorie *ignored* - das sind alle Felder, die nicht in einer der beiden anderen Kategorien sind - werden nicht verarbeitet (also nicht im Literaturverzeichnis angezeigt). Diese Felder eignen sich etwa für eigene Kommentare oder die Angabe von Informationen, die von anderen Literaturprogrammen verarbeitet wer-

Typ	Klasse	Felder
article	required	author, title, journal, year
	optional	volume, number, pages, month, note
book	required	author (oder editor), title, publisher, year
	optional	volume, series, address, edition, month, note
booklet	required	title
	optional	author, howpublished, address, month, year, note
misc	required	*Eines der optionalen Felder*
	optional	author, title, howpublished, month, year, note

Tabelle 4.1: Beispiele für verschiedene Typen von Einträgen in **.bib*-Dateien und deren Felder

den. Tabelle 4.1 gibt einige Beispiele und verdeutlicht die Zusammenhänge. Vollständige Listen aller Eintragstypen sind in gängigen LaTeX-Büchern enthalten.

Syntax der Literatureinträge

Die Einträge in den **.bib*-Dateien müssen wie folgt formatiert sein, um von BibTeX verarbeitet werden zu können:

Syntax für Literatureingaben in **.bib*-Dateien

Allgemein	Beispiel
@<*Eingabetyp*>{<*Schlüssel*>,	`@book{kop:96,`
<*Feld_1*>={<*Eintrag_1*>},	`author={Kopka, Helmut},`
<*Feld_2*>={<*Eintrag_2*>},	`title={\LaTeX \,- Einf\"{u}hrung},`
	`edition={second},`
...	`publisher={Addison-Wesley},`
<*Feld_n*>={<*Eintrag_n*>}	`year={1996}`
}	`}`

In der Regel ist die Funktionalität der im Hauptdokument (**.tex*) eingebundenen usepackages auch in den **.bib*-Dateien gegeben. Trotzdem sollten in den **.bib*-Dateien Umlaute und Sonderzeichen in LaTeX-Syntax eingegeben werden (etwa `\"{a}` für *ä* usw.),

damit die Literaturdateien geräteunabhängig benutzt werden können. Um von BibTEX erkannt werden zu können, müssen alle Literaturdateien im Arbeitsordner liegen.

Regeln für die Syntax der Feldeinträge

Für die Eingabe der Literaturinformationen in die Felder gelten einige Regeln, die in LaTeX-Standardwerken aufgeführt sind. Im Folgenden wird auf einige Richtlinien zur Eingabe des Autors eingegangen.
Die Eingabe des Autors kann nach dem Schema <*Nachname*>, <*Vorname*> oder <*Vorname*> <*Nachname*> erfolgen. `{Helmut Kopka}` führt folglich zu gleichem Resultat wie `{Kopka, Helmut}` (s. o.). Werden Vor- und Nachname ohne Komma getrennt geschrieben (also in der Form <*Vorname*> <*Nachname*>), so interpretiert BibTEX das letzte Wort als Nachnamen. Demzufolge ist `{Hans Dietrich Genscher}` eine korrekte Eingabe, i. e. die beiden ersten Wörter werden als Vornamen interpretiert (`{Genscher, Hans Dietrich}` ist ebenfalls korrekt.). Bei mehr als einem Nachnamen führt nur das Format <*Nachname*>, <*Vorname*> zum richtigen Ergebnis: Da bei der Schreibweise ohne Komma immer das letzte Wort als Nachname interpretiert wird (s. o.), würde bei `{Doris Schr\"{o}der K\"{o}pf}` fälschlicherweise *Schröder* als (zweiter) Vorname interpretiert. `{Schr\"{o}der K\"{o}pf, Doris}` ist hier richtig.
Kleingeschriebene Wörter werden im Feld *author* als Hilfswörter interpretiert (und bei der alphabetischen Anordnung nicht berücksichtigt). Beispiel: `{Ludwig van Beethoven}`. Ist dies nicht erwünscht, kann durch `{Ludwig {van Beethoven}}` erzwungen werden, dass der Eintrag bei *V* eingeordnet wird.
Mehrere Namen werden im *author*-Feld durch *and* getrennt; BibTEX setzt hier ein Komma ein. Falls *and* Teil des Namens ist und ausgeschrieben werden soll, geschieht dies wieder durch Umklammern des Wortes: `{{Black and Decker}}`. „Et al.“ wird durch Eingabe von `{and others}` erzeugt: `{Kopka and others}`.

Hinweis zur Groß-/Kleinschreibung

Die Groß-/Kleinschreibung im Feld *title* wird von BibTEX übernommen: Werden alle Wörter im Titel groß geschrieben, so erscheinen sie mit Ausnahme von Konjunktionen und Präpositionen auch im Literaturverzeichnis groß, wenn es sich beispielsweise um den Eingabetyp *book* handelt. Beim Eingabetyp *article* sorgt BibTEX dafür, dass

in der Bibliographie alle Wörter des Titels außer dem ersten klein geschrieben werden. Dies entspricht der Konvention im englischsprachigen Raum und ist im Deutschen *nicht* erwünscht, weil Substantive im Titel deutscher Zeitschriftenartikel groß geschrieben werden. Durch Umklammern der Großbuchstaben wird dies erzwungen: `{\"{U}ber {M}a\ss{}e der praktischen {S}ignifikanz}`.

Editoren für Literaturdateien

Da die Eingabe der Literaturstellen in der Syntax der **.bib*-Dateien relativ umständlich ist, stehen einige mehr oder weniger brauchbare Editoren für diese Arbeit zur Verfügung. *BibDesk* für Mac OS X bietet eine sehr komfortable Lösung, *BibView* wurde für UNIX entwickelt. Unter Windows ist wohl das Java-basierte *JabRef* gegenüber dem Programm *wbibdb* die bessere Alternative.
Davon abgesehen bieten einige der LaTeX-Editoren/ Entwicklungsumgebungen Unterstützung bei der Eingabe der Literatur an. So sind beispielsweise in *Eclipse* Formatvorlagen für die gängigsten Eintragstypen vorhanden, die sich als sehr hilfreich bei der Literatureingabe erweisen.

4.4.4 Style-Dateien (*.bst)

Style-Dateien legen fest, *wie* die Literaturangaben von BibTeX zur Erstellung der Bibliographie verarbeitet werden, i. e. wie das Literaturverzeichnis formatiert wird. BibTeX-Style-Dateien haben die Erweiterung *.bst* und werden mit dem Befehl `\bibliographystyle{}` eingebunden, wobei in den geschweiften Klammern der Name der Datei (ohne Dateierweiterung) übergeben wird (s. o.). Die folgenden vier Vorlagen sind in jeder LaTeX-Distribution enthalten und sollten für gewöhnliche Zwecke ausreichen[Kop96]:

plain.bst Die Eintragungen im Literaturverzeichnis erfolgen nach den alphabetisch geordneten Autorennamen. Die einzelnen Eintragungen erhalten als Kennzeichnung (in der Bibliographie) laufende Nummern in eckigen Klammern, mit 1 beginnend. Diese Kennzeichnung erscheint im laufenden Text an der Stelle, an der im Quelltext ein entsprechender `\cite{}`-Befehl steht.

unsrt.bst Die Eintragungen im Literaturverzeichnis erfolgen in der Reihenfolge der

`\cite{}`- und `\nocite{}`-Befehle. Ansonsten erfolgt die Kennzeichnung und Anordnung wie beim Stil *plain*.

alpha.bst Die Anordnung erfolgt wie bei *plain*, die Kennzeichnung jedoch nicht in Form einer laufenden Nummer, sodern durch eine Abkürzung des Autorennamens, gefolgt von der Jahreszahl der Veröffentlichung (z.B. Kop96).

abrv.bst Die Anordnung erfolgt wie bei *plain*, die Eintragungen im Literaturverzeichnis sind jedoch kompakter, da Vornamen, Monatsnamen und Journalnamen abgekürzt werden.

Zudem finden sich im Internet zahlreiche **.bst*-Vorlagen, womit sich unzählige Möglichkeiten der Formatierung von Literaturverzeichnis und Verweisen (im Text) realisieren lassen. Darüberhinaus können die Dateien editiert bzw. selbst programmiert werden[Pat88b, Rai]. Obwohl die **.bst*-Dateien in einer einfachen stackbasierten Sprache geschrieben sind, die mit 10 Befehlen auskommt, ist ihre Programmierung nicht ganz unkompliziert. Zur Vereinfachung steht die Datei *makebst.tex* zur Verfügung. Bei ihrer Kompilierung (`latex makebst`) startet sie einen Abfragemodus, wodurch Schritt für Schritt eine nach den Wünschen des Benutzers angepasste Style-Datei erzeugt wird.

4.4.5 Erweiterungen

Die Funktionalität von BibTEX lässt sich vielfältig erweitern. Dies geschieht LATEX-typisch anhand von *usepackages*, die in das Quelldokument eingebunden werden. In diesem Abschnitt werden beispielhaft zwei *usepackages* vorgestellt, die von besonderer Bedeutung für „collaborative writing“ bzw. Erstellung von Büchern sind und die Erzeugung mehrerer Bibliographien ermöglichen.

Aufteilung der Bibliograpie mit dem *multibib*-Package

Mit dem *multibib*-Package ist es möglich, das Literaturverzeichnis in einem Dokument in mehrere Teile zu untergliedern - z.B. thematisch. Dazu muss die Datei *multibib.sty* in der Präambel des Quelltextes eingebunden werden (`\usepackage{multibib}`). Tabelle 4.2 zeigt die für das Packet relevanten Befehle.

Befehl	Beschreibung
`\newcites{<Suffix>}{<Name>}` (*in der Präambel*)	Definiert eine neue Gruppe von Befehlen, die sich aus der üblichen BibTEX-Syntax durch anhängen von *<Suffix>* bilden lassen und sich auf das Literaturverzeichnis mit der Überschrift *<Name>* beziehen. Beispiel: `\newcites{deu}{Deutsche Literatur}`
`\bibliography<Suffix>{<Datei(en)>}`	Erzeugt ein (zusätzliches) Literaturverzeichnis, auf das sich alle Befehle mit dem *<Suffix>* beziehen (s. o.). Hierfür werden die **.bib*-Dateien *<Datei(en)>* eingebunden. Beispiel: `\bibliographydeu{litdeu1,litdeu2}`
`\bibliographystyle<Suffix>{<Datei>}`	s. o.
`\cite<Suffix>{<Schlüssel>}`	
`\nocite<Suffix>{<Schlüssel>}`	

Tabelle 4.2: Befehle für *multibib*

```
\documentclass{article}
\usepackage{multibib}
\newcites{ltex}{\TeX\ and \LaTeX\ References}

\begin{document}
References to the \TeX book \citeltex{Knuth:1991}
and to Lamport's \LaTeX\ book, which appears
only in the references\nociteltex{Lamport:1994}.
Finally a cite to a Postscript tutorial
\cite{Adobe:1985}.

\bibliographystyleltex{alpha}
\bibliographyltex{lit}

\renewcommand{\refname}{Postscript References}
\bibliographystyle{plain}
\bibliography{lit}

\end{document}
```

References to the TEXbook [Knu91] and to Lamport's LATEX book, which appears only in the references. Finally a cite to a Postscript tutorial [1].

TEX and LATEX References

[Knu91] Donald E. Knuth. *The TEX book*. Addison-Wesley, Reading, Massachusetts, 1991.

[Lam94] Leslie Lamport. *LATEX: A Document Preparation System*. Addison-Wesley, Reading, Massachusetts, 2 edition, 1994.

Postscript References

[1] Adobe System Incorporated. *Postscript Language Tutorial and Cookbook*. Addison-Wesley, Reading, Massachusetts, 1985.

Abbildung 4.2: Beispiel

Das grundsätzliche Prinzip besteht darin, neue Literaturverzeichnisse hinzuzufügen und ihnen ein Suffix zuzuordnen. Wird dann an die üblichen BibTEX-Befehle das entsprechende Suffix angehängt, so beziehen sie sich auf das jeweilige Literaturverzeichnis. Das Beispiel in Abbildung 4.2 ist dem *multibib*-Manual [Han04b] entnommen.

Mehrere Literaturverzeichnisse mit dem *bibunits*-Package

Unter Umständen ist es erwünscht, mehrere Literaturverzeichnisse in einer Datei einzubinden. Dies ist in der Regel bei Büchern der Fall, wo etwa für jedes Kapitel ein eigenes Verzeichnis erzeugt werden soll. Mit dem *bibunits*-Package wird dies erreicht, indem entweder in der Quelldatei mehrere *„bibunit“*-Umgebungen definiert werden, für die jeweils eine eigene Bibliographie angelegt wird, oder automatisch für bestimmte vordefinierte Umgebungen (z.B. *chapter*-Umgebungen) je ein Verzeichnis erstellt wird. Zusätzlich kann es ein Gesamtverzeichnis geben.

Unterteilung mit *bibunit*-Umgebungen

Mit den Befehlen `\begin{bibunit}[<*.bst>]` ... `\end{bibunit}` können *bibunit*-Umgebungen erzeugt werden, wobei *<*.bst>* der Name der Style-Datei für die zugehörige Bibliographie ist (ohne Dateierweiterung). Innerhalb einer *bibunit*-Umgebung kann mit dem Befehl `\putbib[<*.bib>]` eine Bibliographie erstellt werden; *<*.bib>* spezifiziert

dabei die einzubindende(n) Literaturdatei(en). Alle `\cite{}`- und `\nocite{}`-Befehle, die in einer *bibunit*-Umgebung erscheinen, beziehen sich auf die so erzeugte Bibliographie. Das Beispiel in Abb. 4.3 verdeutlicht den Umgang mit den Befehlen.

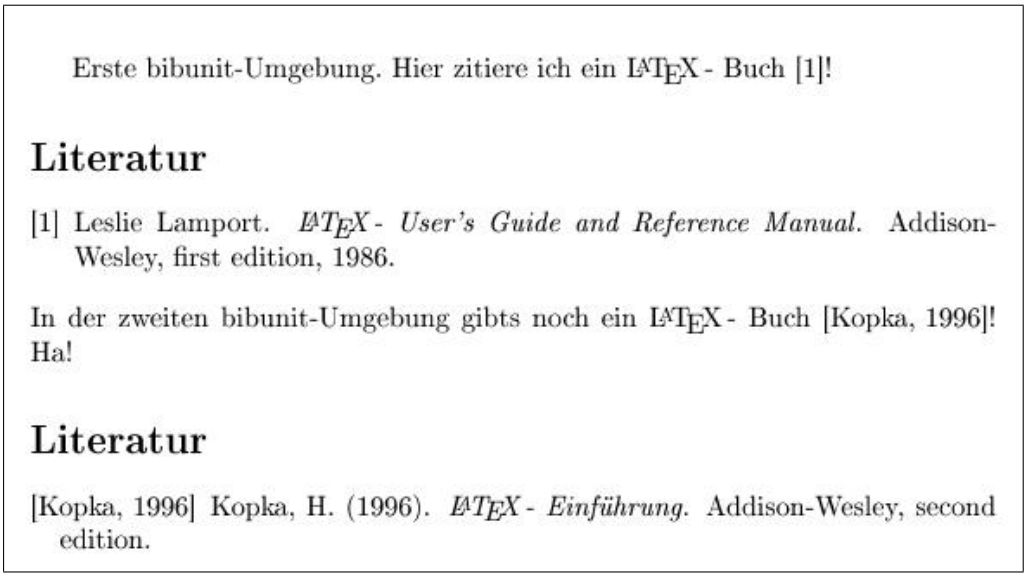
```
\documentclass{article}
\usepackage{bibunits}
...
\begin{document}
%1. bibunit-Umgebung
\begin{bibunit}[plain]
Erste bibunit-Umgebung. Hier zitiere ich ein \LaTeX \,-
Buch \cite{lam:86}!
\putbib[bibunits-bsp1]
\end{bibunit}
%2. bibunit-Umgebung
\begin{bibunit}[apalike]
In der zweiten bibunit-Umgebung gibts noch ein \LaTeX \,-
Buch \cite{kop:96}! Ha!
\putbib[bibunits-bsp2]
\end{bibunit}
\end{document}
```

Abbildung 4.3: Beispiel

Unterteilung mit *chapter*- bzw. *section*-Umgebungen

Alternativ zur Unterteilung des Dokuments in *bibunit*-Umgebungen kann auch automatisch für jede *chapter*- oder *section*-Einheit des Quelldokuments ein eigenes Literaturverzeichnis erstellt werden. Dies wird durch den Befehl `\bibliographyunit[<Gliederung>]` erreicht, der nach `\begin{document}` stehen muss. Für *<Gliederung>* wird hierbei `\chapter` oder `\section` eingesetzt. Im Beispiel in Abbildung 4.5 werden mehrere Bibliographien anhand der *section*-Umgebungen des Dokuments erzeugt. Eine genaue Beschreibung aller Befehle dieser Option sowie weiterer Möglichkeiten findet sich im *bibunit*-Manual[Han04a].

Globale Bibliographien



Erste bibunit-Umgebung. Hier zitiere ich ein LaTeX - Buch [1]!

Literatur

[1] Leslie Lamport. *LaTeX - User's Guide and Reference Manual.* Addison-Wesley, first edition, 1986.

In der zweiten bibunit-Umgebung gibts noch ein LaTeX - Buch [Kopka, 1996]! Ha!

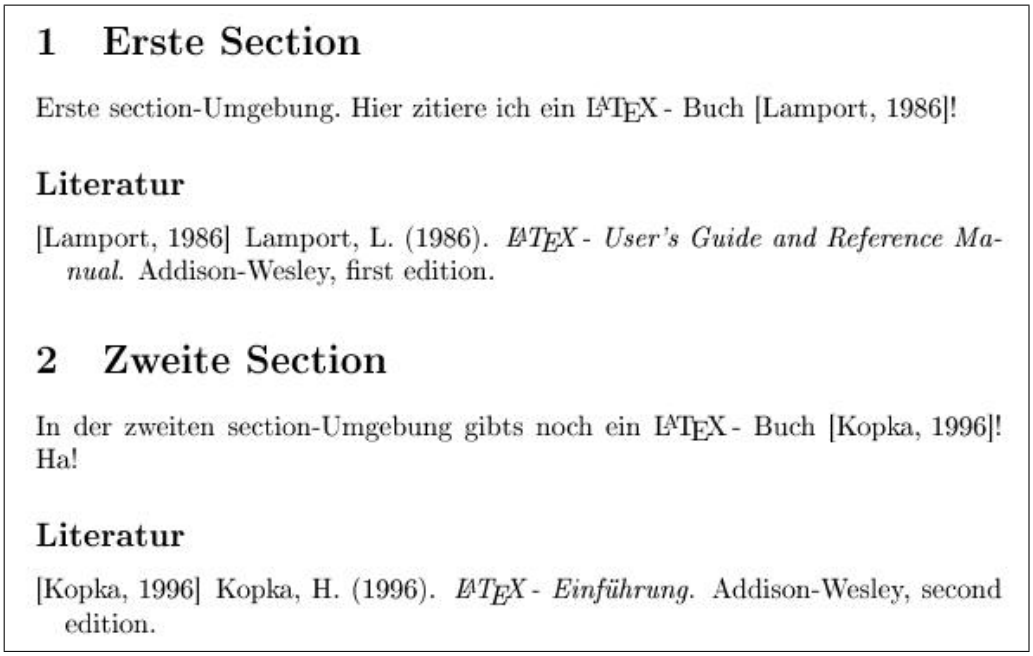
Literatur

[Kopka, 1996] Kopka, H. (1996). *LaTeX - Einführung.* Addison-Wesley, second edition.

Abbildung 4.4: Ausgabe zum Beispiel aus der vorangegengenen Abb.

```
\documentclass{article}
\usepackage[subsectionbib]{bibunits}
...
\begin{document}
\bibliographyunit[\section]
\defaultbibliography{bibunits-bsp1,bibunits-bsp2}
\defaultbibliographystyle{apalike}
\section{Erste Section}
Erste section-Umgebung. Hier zitiere ich ein \LaTeX \,- Buch
\cite{lam:86}!
\putbib
\section{Zweite Section}
In der zweiten section-Umgebung gibts noch ein \LaTeX \,- Buch
\cite{kop:96}! Ha!
\putbib
\end{document}
```

Abbildung 4.5: Beispiel



1 Erste Section

Erste section-Umgebung. Hier zitiere ich ein LaTeX - Buch [Lamport, 1986]!

Literatur

[Lamport, 1986] Lamport, L. (1986). *LaTeX - User's Guide and Reference Manual.* Addison-Wesley, first edition.

2 Zweite Section

In der zweiten section-Umgebung gibts noch ein LaTeX - Buch [Kopka, 1996]! Ha!

Literatur

[Kopka, 1996] Kopka, H. (1996). *LaTeX - Einführung.* Addison-Wesley, second edition.

Abbildung 4.6: Ausgabe zum Beispiel aus der vorangegengenen Abb.

Wenn pro Kapitel oder Abschnitt ein separates Literaturverzeichnis besteht, ist es oft erwünscht, eine weitere, globale Bibliographie einzubinden - die beispielsweise am Ende eines Buches alle Literaturangaben der einzelnen Verzeichnisse zusammenfasst. Dies geschieht mit den gewohnten Befehlen (`\bibliography{}, \bibliographystyle{}`). Innerhalb von *bibunit*-Umgebungen kann durch anhängen eines * an die Zitierbefehle (also `\cite*{...}` bzw. `\nocite*{...}`) erreicht werden, dass die jeweiligen Literaturstellen nicht nur im „lokalen“, sondern auch im globalen Verzeichnis erscheinen.

Kompilieren von Quelldokumenten unter Einbindung von *bibunit*

Für das Erzeugen der Ausgabedatei aus der Kommandozeile gelten besondere Regeln, falls *bibunits* als usepackage im Quelldokument geladen ist:
Ein LaTeX-Durchlauf erzeugt für jede *bibunit*-Umgebung eine eigene **.aux*-Datei. Diese Dateien werden *bu<i>.aux* mit <i> als Laufindex benannt. Jede dieser Dateien muss von BibTEX kompiliert werden, sodass z.B. bei 3 *bibunit*-Umgebungen folgende Kommandos eingegeben werden müssen, um das Dokument fehlerfrei zu kompilieren (mit *document.tex* als Eingabedatei):

latex document
bibtex bu1
bibtex bu2
bibtex bu3
latex document
latex document

4.5 JabRef – Ein BibTEX-Editor

Moritz Gerner-Beuerle

Die Idee von BibTEX, eine zentrale Verwaltungsdatei aufzubauen, aus der man seine wissenschaftlichen Arbeiten bequem mit einem Literaturverzeichnis versorgen kann, findet seine Grenzen in Übersicht und Komfortabilität, wenn man eine große Anzahl von Literatur verwalten möchte.

JabRef ist ein Betriebssystem-unabhängiges Programm, da es in Java programmiert wurde. Ohne Schwierigkeiten lassen sich *.bib-Dateien importieren und bearbeiten. Such- und Sortierfunktionen helfen beim Suchen nach einem bestimmten Eintrag. Ein weiterer Vorteil von JabRef ist die Konfigurierbarkeit. So lassen sich eigene Eintragstypen definieren. Außerdem lassen sich die Eintragsfelder ergänzen. So ist es etwa denkbar, sich ein Eintragsfeld für den Standort oder die Signatur eines Bibliotheksbuches zu definieren. JabRef unterstützt das ergänzen der Literturangaben über die CiteSeer-Internetdatenbank, in der Literaturangaben zu wissenschaftlichen Arbeiten gesammelt sind.

Es gibt noch einige andere Features, die das Arbeiten mir LATEXund BibTEX vereinfachen, wie zum Beispiel die Ausgabe der Literaturdatenbank in verschieden Formate. Was JabRef aber im wissenschaftlichen Betrieb zu einem geeigneten Werkzeug auch für Benutzer anderer Textverarbeitungsprogramme macht, ist die Möglichkeit, sich eine flexible und umfangreich konfigurierbare Literaturdatenbank zu schaffen.

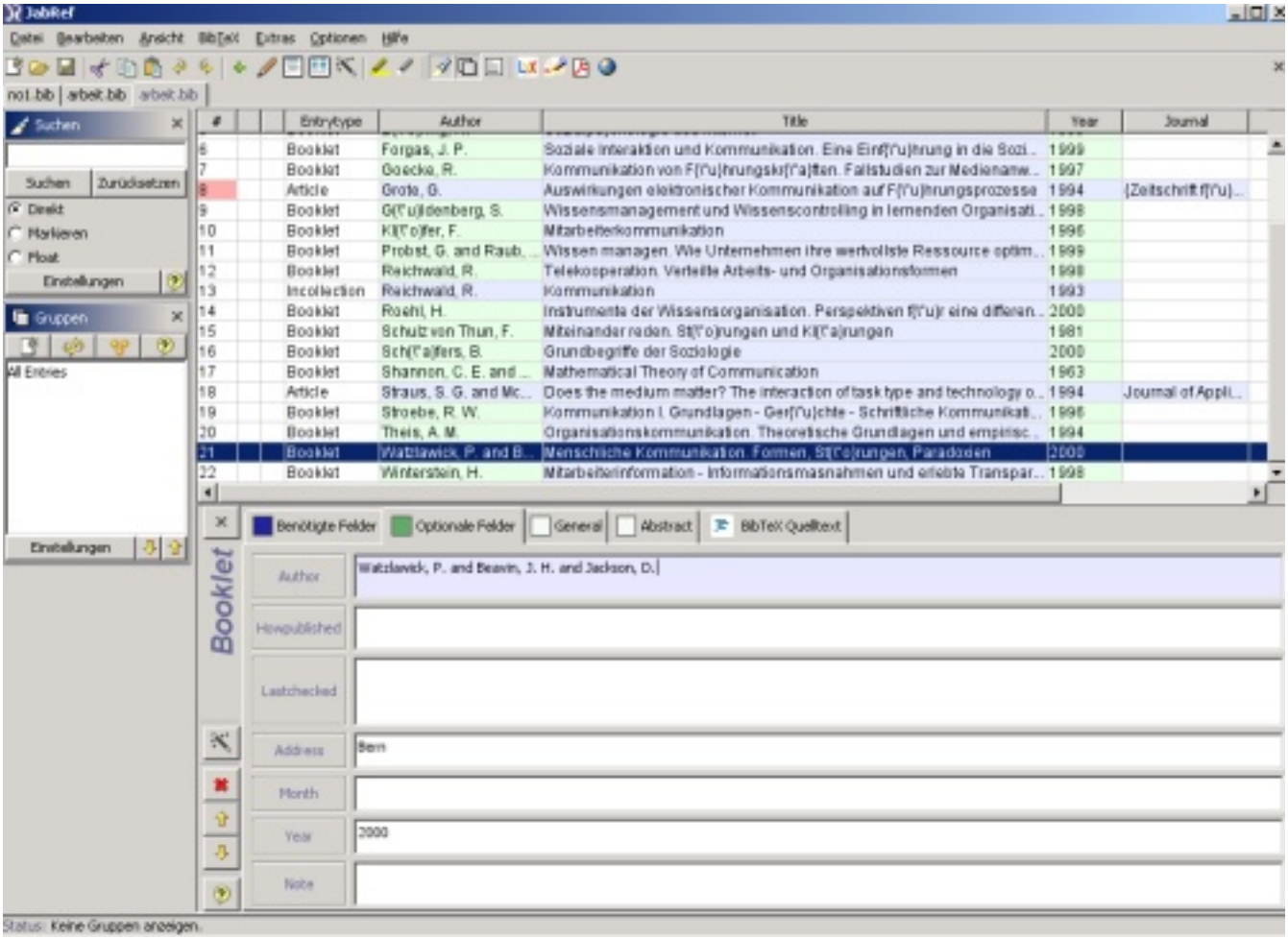


Abbildung 4.7: JabRef Hauptfenster *http://jabref.sourceforge.net*

4.6 Indexerstellung

Nicolas Paul

4.6.1 Das Stichwortverzeichnis

Dieser Abschnitt beginnt mit einigen, allgemein gehaltenen, Informationen zum Thema Stichwortverzeichnis.

Allgemeines

Ein Stichwortverzeichnis ist ein alphabetisches Verzeichnis von Schlüsselbegriffen (Namens- und Sachbezeichnungen) mit dem Verweis auf Seitenzahlen in einem Buch.

Stichwortverzeichnisse in Büchern (oder Fachzeitschriften) dienen dazu, dem Leser das Werk, durch das Nachschlagen von Begriffen, schnell zu erschließen.

Meist befindet sich das Stichwortverzeichnis am Ende eines Buches. Im Normalfall hat es eine alphabetische Ordnung. Häufig bestehen Stichwortverzeichnisse aus einem Hauptstichwort, dem verschiedene Unterstichworte zugeordnet sind. Ergänzt wird das jeweilige Stichwort durch die jeweils dazu angegebene Fundstelle. Dies ist jeweils die Seite oder der durch eine Randziffer gekennzeichnete Textabschnitt, wo das genannte Stichwort vorkommt.

Die Problematik welche Wörter überhaupt in den Index aufgenommen werden, muss ein Autor selbst lösen. Da diese Aufgabe genaue Kenntnisse des Inhaltes erfordert und der Index auch auf Schwerpunkte innerhalb des Werkes hinweisen soll, ist es am sinnvollsten wenn der Autor selbst entscheidet was in den Index aufgenommen wird.

4.6.2 Das Paket *makeidx*

In diesem Abschnitt wird das Paket *makeidx* vorgestellt. *makeidx* ist nicht das einzige Paket zur Indexerstellung, es beinhaltet allerdings alle wichtigen Funktionen und ist gut dokumentiert.

Indexerstellung

Als erstes muss das Paket mit dem Befehl *usepackage{}* in das dokument eingebunden werden. Der Befehl *makeindex* erstellt den Index. Hierbei ist zu beachten, dass *makeindex* in der Päambel, also vor dem Beginn des Dokumentes, steht. *printindex* steht an der Stelle im Dokument, wo der Index später erscheinen soll. (Siehe Abbildung 4.8)

```
\usepackage{makeidx} %Bereitstellung des Paketes

\makeindex %Der Index wird erstellt

\begin{document}

\printindex %Die Stelle wo der Index erscheinen soll

\end{document}
```

Abbildung 4.8: Einbindung in das LaTeX Dokument

Begriffe werden mit dem Befehl **index{}** markiert und somit in den Index aufgenommen. Man schreibt den Befehl einfach hinter das zu indizierende Wort, fügt in die Klammern das Stichwort ein und es wird standardmäßig ein Indexeintrag aus Stichwort und Seitenzahl erzeugt.[Los05]

```
\index{Stichwort}
```

Weitere Optionen füer den \index{}-Befehl:

- **Unterstichworte:**

  Man hat die Möglichkeit ein Stichwort in bis zu 3 Ebenen aufzuteilen. Die Ebenen werden dabei mit dem `!`-Zeichen voneinander getrennt.

  - `\index{Stichwort ! Unterschtichwort ! Unterunterstichwort}`

- **Sonderzeichen:**

  Möchte man ein Sonderzeichen in den Index aufnehmen und es beispielsweise nach seinem Namen in den Index einsortieren, geht das so:

– `\index{Sortierung @ Darstellung}`

- **Bereiche:**

  Manchmal möchte man bestimmten Indexbegriffen grö/ssere Bereiche zuordnen. So dass im Index dann nicht jede einzelne Seite, sondern bspw. Seite 30-35 steht. Außerdem müssen die Stichwörter in dem Bereich nicht einzeln markiert werden. Am Anfang des Bereichs muss `\index{Stichwort|(}` und am Ende `\index{Stichwort|)}` stehen.

- **Verweise:**

  Häufig finden sich in Stichwortverzeichnissen auch Verweise auf andere Begriffe. Verweise auf synonyme Begriffe erleichtern dem Nutzer das Finden. (Beispiel: Kommando: siehe Befehl)

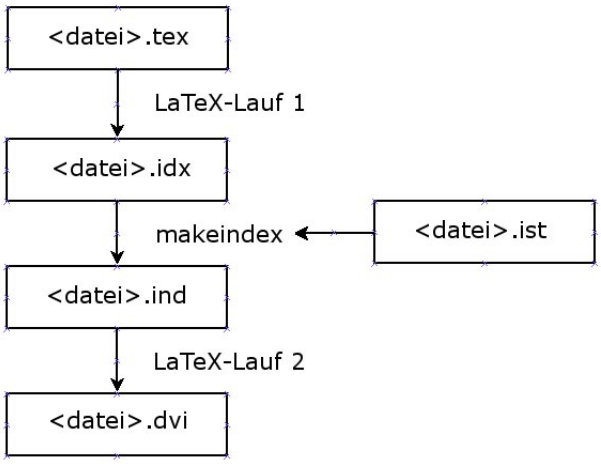
  – `\index{Kommando | see{Befehl}}`

  Das Zeichen | legt fest, dass ein Befehl folgt. Weiterhin gibt es siehe-auch Verweise. Diese weisen auf ähnliche, sachlich über- oder untergeordnete oder auf weiterführende Begriffe hin. (Beispiel: Deutsche Bahn AG: siehe auch Reichsbahn)

  – `\index{Deutsche Bahn AG | seealso{Reichsbahn}}`

Indexformatierung

Um dem Index ein optisch ansprechendes Format zu geben, müssen wir uns zunächst anschauen, wie LaTeX den Index erstellt.[NN03]



Schritt 1:

Latex erzeugt beim erstem Durchlauf eine **.idx** - Datei, welche alle markierten Begriffe in der Reihenfolge ihres Auftretens enthält.

Schritt 2:

Der Befehl `\makeindex` sortiert die Einträge der **.idx** - Datei und erstellt eine Datei mit der Endung **.ind**, welche den fertigen Latex Quelltext des Index enthält (in einer theindex-Umgebung). Das Layout des Index wird separat über eine Stildatei (**.ist**) bestimmt.

Schritt 3:

Beim nochmaligen Durchlauf von Latex wird die **.idn** - Datei an der Stelle des Quelltextes, wo der Befehl `\printindex` steht, eingebunden.

Bei der Indexerstellung wird also das Hilfsprogramm `makeindex` aufgerufen. Dessen Aufruf kann mit einer Menge optionaler Parameter versehen werden. Einige wichtige werden hier aufgeführt.

- **-l**

  Leerzeichen bleiben beim Sortiervorgang unberücksichtigt.

- **-c**

  Führende oder angehängte Leerzeichen werden ignoriert.

- **-g**

  Sortiert nach DIN 2007: Symbole; Kleinbuchstaben; Großbuchstaben; Zahlen

- **-s dateiname.ist**

  Die angegebene Stildatei sorgt für die Formatierung des Indexes.

- Ein Beispielaufruf: `makeindex -g -s stildatei.ist indextest`

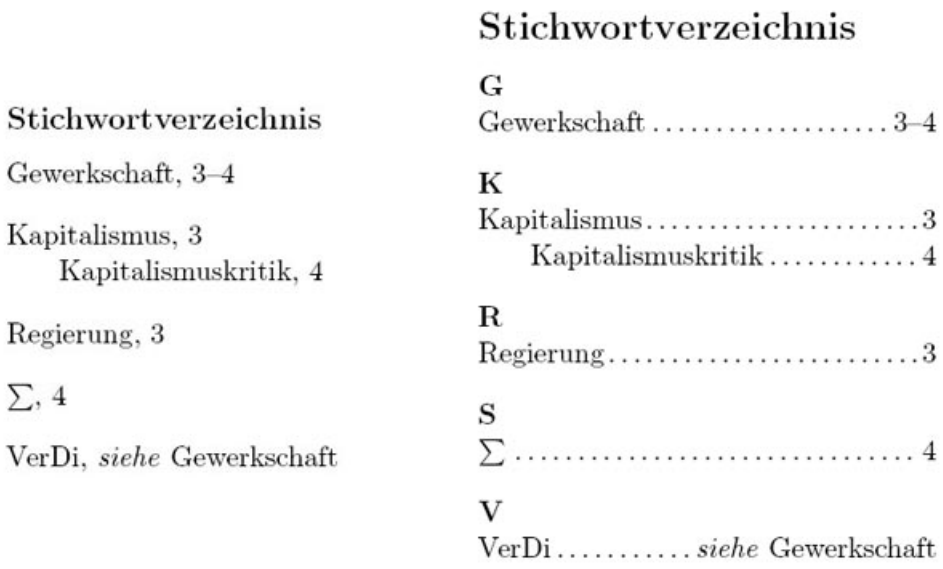
Die Stildatei legt das Aussehen des Index fest. Sie muss separat erstellt werden und mit **.ist** enden. Es stehen wiederum jede Menge Befehle zur Verfügung. Das Beispiel in Abbildung 4.9 soll dieses verdeutlichen.

```
group_skip "\n\n \\indexspace\n"

headings_flag 1
heading_prefix "{\\bf"
heading_suffix "}"

delim_0 "\\dotfill"
delim_1 "\\dotfill"
delim_2 "\\dotfill"
```

Abbildung 4.9: Stildatei.ist [Ste05]

Das Kommando `headings_flag 1` führt dazu, dass die einzelnen Buchstabengruppen ihren Buchstaben, großgeschrieben, als Überschrift bekommen. Mit `-1` gibt's den Buchstaben klein geschrieben. `delim_0 "\\dotfill"` füllt den Bereich zwischen einem Eintrag der Hauptebene und der Seitenzahl mit Punkten aus. Mit `delim_1` und `delim_2` gilt das gleiche für Unter und Unterunterebene. Normalerweise wird der Index zweispaltig dargestellt. Möchte man dies ändern, so kann man mit dem multicol-Paket weitere Formatierungen vornehmen.

Weitere Optionen:



Stichwortverzeichnis

Gewerkschaft, 3–4

Kapitalismus, 3
Kapitalismuskritik, 4

Regierung, 3

$\sum$, 4

VerDi, *siehe* Gewerkschaft

Stichwortverzeichnis

G
Gewerkschaft 3–4

K
Kapitalismus 3
Kapitalismuskritik 4

R
Regierung 3

S
$\sum$ 4

V
VerDi ..... *siehe* Gewerkschaft

Abbildung 4.10: Vergleich, ohne - und mit Stildatei

`\renewcommand{\indexname}{Stichwortverzeichnis}` –> Die Überschrift Index wird durch die Überschrift Stichwortverzeichnis ersetzt.

`\addcontentsline{toc}{section}{Stichwortverzeichnis}` –> Stichwortverzeichnis taucht im Inhaltsverzeichnis auf und wird wie eine normale Überschrift behandelt.[Ste05]

4.6.3 Das Glossar

Mit dem Paket *glossar* kann man einem Text ein Glossar hinzufügen.[NN03] Aber was ist eigentlich ein Glossar?

Allgemeines

Im Gegensatz zu einem Wörterbuch enthält ein Glossar vorrangig Begriffe und Definitionen. Die Definitionen in einem Glossar sollen anders als in größeren Nachschlagewerken wie etwa einem Lexikon möglichst kurz und eindeutig sein. Die Grenze zwischen Glossar und Lexikon ist jedoch fließend, so werden manche Glossare auch als Definitionslexikon bezeichnet.

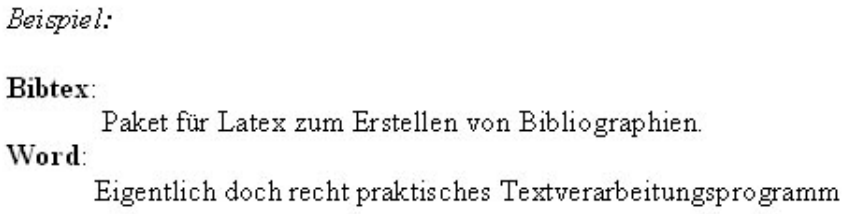


Abbildung 4.11: So könnte ein Glossar aussehen

glossar.sty

Um ein Glossar in ein Latex-Dokument einzubinden steht das Paket *glossar* zu Verfügung. Ähnlich wie bei der Indexerstellung, werden die Glossareinträge in eine externe Datei, mit der Endung **.glo**, geschrieben. Diese werden dann mit dem *makeindex* Programm sortiert, in eine **.gls** - Datei (in eine description-Umgebung) geschrieben und in das Dokument integriert.

```
\usepackage{glossar}

\makeglossary %schreibt die Glossareintraege in .glo Datei

\glentry{Stichwort}{Erklärungstext} %Glossareintrag

\printglossary %Bindet die .gls Datei ein
```

Abbildung 4.12: Einbindung des Glossars

Beim Aufruf von *makeindex* wird wieder eine stildatei eingebunden. Der Aufruf lautet:

```
makeindex  bsp_glossar.glo -s bsp_glossar.ist -o bsp_glossar.gls
```

Wichtig hierbei ist, dass der Parameter **-o** die Ausgabedatei angibt, welche die Endung **.gls** haben muss.

Kapitel 5

Was kann bei der Erstellung von Dokumenten mit LaTeX helfen

Sascha Purmann, Dennis Tomas

5.1 Einleitung

Unix versus Windows, LaTeX versus Word. Eigentlich ist die Sache ja klar: Unix ist besser als Windows und LaTeX ist besser als Word. Trotzdem hat LaTeX nicht nur sonnige Seiten. Zum Beispiel muss man die LaTeX-Befehle lernen, weshalb man nicht ohne Weiteres sofort mit LaTeX loslegen kann. Die Lernkurve verläuft weniger steil, als man das gerne hätte und wer nicht regelmäßig mit LaTeX arbeitet, vegißt schon mal den ein oder anderen Befehl. Im Allgemeinen wird man wohl ein LaTeXbuch als Gedächtnisstütze neben seinem Computer liegen haben, doch das ist nicht das Einzige, was man für sich und seinen Erfolg mit LaTeX tun kann. Dieses Kapitel beschreibt ein paar Hilfsprogramme, die den Umgang mit LaTeX erleichtern sollen.

Führen wir uns noch einmal vor Augen, wie man ein Dokument mit LaTeX erzeugt. Erst erzeugt man mit einem Texteditor eine Eingabedatei, die den Text eingebettet in LaTeX-Befehle enthält. Diese Datei wird LaTeX als Argument übergeben. LaTeX erzeugt dann eine Datei, die den gesetzten Text in einem geräteunabhängigen Format (DVI, PDF oder PostScript) enthält. Diese Datei kann man sich in einem Previewer anschauen und wenn gewünscht ausdrucken. Bleibt eigentlich nichts zu wünschen übrig, oder doch?

5.2 Unix-Philosophen oder „Small is beautiful“

In der Art und Weise, wie LaTeX arbeitet, folgt es einer Unix-Philosophie. Wer das erste Mal mit Unix zu tun hat, ist meist irritiert durch die vielen Kommandos, die anscheinend für die tägliche Bedienung benötigt werden. Es gibt eine Menge guter Bücher, die beim Linuxeinstieg helfen, die allerdings auch nicht ganz billig sind. Aber auch ohne eine private Linux-Bibliothek muss der Einstieg keine große Hürde sein. Zum einen kann man sich in der Lehrbuchsammlung oder der Informatik-Bibliothek die Standardwerke ausleihen, zum anderen wimmelt es im Internet von guten Linux-Einführungen.

Unixe warten mit vielen kleinen Anwendungen auf (oft nur ein paar Zeilen Quellcode lang), unter Windows kennt man hingegen eher Mammutanwendungen. Diese sollen dann auch alles können. Windows kann deshalb aber noch lange nicht mehr als Unix. Erstens sind die Mammutanwendungen in nichts wirklich richtig gut – Microsoft Word ist nur ein Beispiel – und zweitens kann man unter Unix die verschiedenen kleinen Programme miteinander verbinden und so auch komplexere Anforderungen erfüllen. Die Ausgabe des einen Programms kann zur Eingabe eines anderen Programmes werden, und so weiter. Man kann sich das so vorstellen, als ob ein Profi seine Arbeit erledigt und das Produkt dann einem Profi einer anderen Sparte übergibt. Zur Illustration: Ein Haus wird auch nicht von einem Handwerker allein gebaut, sondern entsteht unter der Mitwirkung verschiedener Handwerker, und man wird sicher nicht von einem Maurer verlangen, die Elektroinstallation zu machen. Unixe realisieren quasi dass, was wir als Arbeitsteilung bezeichnen. Da die Verbindung von Programmen durch die Shell erledigt wird nennt man das Ganze auch Shellprogrammierung. Eine besondere Voraussetzung für die reibungslose Zusammenarbeit von Programmen ist, dass das eine Programm etwas produziert, das von dem anderen Programm verstanden wird. Unter Unix sind alle Dateien Textdateien. Die Shellprogrammierung stellt zweifelsohne Modularität dar, denn es werden autarke Softwarekomponenten an verschiedenen Stellen benutzt. Mit etwas Kreativität kann man so unter Unix viele Probleme selbst lösen.

Ein Vergleich zwischen Windows und Linux läßt sich aber auch anders ziehen. Es geht nämlich auch um Sprache: Während man unter Windows Betriebssystemen erst auf das eine und dann auf das andere „zeigt“ und dann aus einem Menü auswählt, um etwas mit zwei Dingen zu tun, „spricht“ man mit seinem Unix Betriebssystem und „sagt“ ihm dass es mit diesem oder jenem und einem anderen dieses oder jenes machen soll.

Dem kindlichen „da“ von Windows steht die erwachsene Kommandosprache von Unix gegenüber.

5.3 Emacs

Der Emacs-Editor ist ein flexibler, das heißt vor allem ein umfassend konfigurierbarer und ein vielfältig erweiterbarer Texteditor. Man kann Emacs für nahezu alles, was man mit dem Computer machen kann, benutzen. So gibt es Makropaktete, um Emacs als News-Reader, Webbrowser oder Email-Programm zu benutzen. Neben Emacs, welches in der Shell benutzt wird, gibt es auch eine entsprechende Version für grafische Benutzeroberflächen, XEmacs.

5.3.1 Installation

Emacs ist für die gängigen Plattformen verfügbar. Auf der Emacs-Homepage[1] läßt sich die jeweils richtige Version plus Installationsanleitung finden.

5.3.2 Bedienung

Die Bedienung von Emacs ist gewöhnungsbedürftig, was jedoch für alle anderen Editoren gilt, mit denen man anfängt zu arbeiten. Es lohnt sich jedoch, zumindest einen Editor gut zu beherrschen, und Emacs ist sicher eine gute Wahl. Er ist auf vielen Systemen, auf denen man arbeitet ohnehin vorhanden oder läßt sich leicht nachinstallieren. Verwendet man Emacs unter einer grafischen Benutzeroberfläche, so ist man schnell verführt, die Maus zu benutzen. Den größten Vorteil hat man jedoch bei ausschließlicher Verwendung der Tastatur, denn Emacs ist vollständig mit Tastaturbefehlen steuerbar. Es gibt zwei Tasten, die dabei wichtig sind: Erstens die Control-Taste und zweitens die Meta-Taste[2] (die entweder die Escape-Taste oder die Alt-Taste ist). Die Control-Taste muss mit einem Buchstaben zusammen gedrückt werden, die Meta-Taste hingegen muss vor einem Buchstaben gedrückt werden. Um ohne Maus auf die Menüleiste von Emacs zuzugreifen, verwendet man `F10`.

[1] siehe Tabelle 5.2 auf Seite 88

[2] für die Control-Taste schreibe ich im Folgenden „Ctrl-“ und für die Meta-Taste „Meta-“

Ctrl-...		
	f	Zeichen vor
	b	Zeichen zurück
	n	in die nächste Zeile
	p	in die vorherige Zeile
	a	an den Zeilenanfang
	e	an das Zeilenende
	v	Seite vor
Meta-...		
	v	Seite zurück
	a	an den Satzanfang
	e	an das Satzende

Tabelle 5.1: Emacs-Befehle: Navigation im Text

Navigation im Text

Als Übungsmaterial lädt man sich mit `Ctrl-x Ctrl-f` unter Angabe des Pfades eine Textdatei. Emacs erscheint zunächts zweigeteilt: Oben sieht man in einem großen Fenster den Text, den man geladen hat und unten sieht man den sogenannten Minibuffer, wo die Befehle, die man eingibt zu sehen sind. Man kann das große Fenster nach belieben unterteilen, um an mehr als einer Sache arbeiten. Es muss jedoch nicht alles, an dem Emacs arbeitet in einem Fenster angezeigt werden – doch dazu später mehr. In einem Text kann man ein Zeichen vor (`Ctrl-f`) oder zurück (`Ctrl-b`) gehen, ein ganzes Wort vor (`Meta-f`) oder zurück (`Meta-b`) gehen oder einen ganzen Satz vor (`Meta-e`) oder zurück (`Meta-a`) gehen. Kleiner Tipp: Damit Emacs einen Satz als Satz erkennt, sind zwei Leerzeichen nach einem Satzendezeichen notwendig, was LaTeX aber bekanntlich nicht stört. Bei längeren Texten besteht Bedarf, ganze Seiten vor (`Ctrl-v`) oder zurück (`Meta-v`) zu gehen oder an den Anfang (`Meta-<`) oder das Ende (`Meta->`) des Dokuments zu springen. Man kann auch „zeilenorientiert“ mit Emacs arbeiten: So bringt einen `Ctrl-n` in die nächste Zeile und `Ctrl-p` in die vorherige Zeile, `Ctrl-e` an das Ende und `Ctrl-a` an den Anfang einer Zeile. Um einen dieser Befehle mehrere Male hintereinander auszuführen kann man den Befehl `Ctrl-u` benutzen: So geht man mit `Ctrl-u 8 Ctrl-f` acht Zeichen vor. Übrigens: Mit `Ctrl-x Ctrl-c` schließt man den Emacseditor und mit `Ctrl-h t` startet man das Turtorial.

Emacs in verschiedenen Modi

Der Emacs-Editor kennt verschiedene Modi, wobei man Haupt- und Untermodi unterscheidet. Ein Modus ist ein Zustand mit bestimmten Voreinstellungen wie farbige Markierung bestimmter Zeichen und veränderte Kurzbefehle. Die Unterschiede in den Hauptmodi sind größer als zwischen den Untermodi, so dass die Untermodi auf jeden Hauptmodus angewendet werden können. Es gibt beispielsweise Hauptmodi für die Bearbeitung von Quellcode verschiedener Programmiersprachen, einen Shellmodus, einen Modus für Texte allgemein und eben auch einen TEXmodus.

Fensterspielchen

Man kann das Hauptfenster weiter aufteilen, so dass man zwei (oder mehr) Emacsprozesse auf einmal betrachten kann. Mit `Ctrl-x 2` teilt man das Fenster horizontal, mit `Ctrl-x 3` vertikal. Zu Beginn haben beide Fenster den gleichen Inhalt, man kann aber nun mit `Ctrl-x Ctrl-f` eine neue Datei in das neue Fenster laden. Um in ein anderes Fenster zu wechseln gibt man den Befehl `Ctrl-x o` ein. So kann man von Fenster zu Fenster springen. Sind viele Fenster geöffnet, so wäre es doch eigentlich besser, wenn man aus einer Auflistung der Fenster auswählen könnte. Zu einer derartigen Auflistung kommt man mit `Ctrl-x b`, allerdings enthält die Auflistung nicht nur die Fenster, sondern alle Emacs-Prozesse, die sogenannten Buffer. Mit `Ctrl-x 0` macht man übrigens die letzte Fensterteilung rückgängig und mit `Ctrl-x 1` schließt man alle Fenster (das heißt jedoch nicht, dass die Emacs-Prozesse beendet wären, was man mit `Ctrl-x b` prüfen kann), bis auf das, welches den Cursor enthält.

5.3.3 Konfiguration

Die Emacs-Einstellungen sind zum einen in einer systemweiten und zum anderen in einer benutzereigenen Konfigurationsdatei gespeichert. Diese Dateien kann man nun entweder per Hand editieren, zum anderen aber auch über die Konfigurationsmenüs. Will man zum Beispiel die Standardschriftgröße ändern, so gibt man folgendes ein, um ins zuständige Konfigurationsmeü zu gelangen:

```
M-x customize-face RETURN default RETURN
```

5.4 Emacs und Auctex

Obwohl Emacs – wie schon erwähnt – von Hause aus über einen LaTeX-Modus verfügt, gibt es weitere Makropakete, die den Umgang mit LaTeX-Dateien vereinfachen. Eines dieser Makropakete heißt Auctex. Trotz der vielen Features bedeutet die Nutzung von Auctex nicht, dass man stundenlang „Neues“ lernen muss. Das Lernen passiert viel mehr by doing, denn man kann wie gewohnt LaTeX- Dokumente bearbeiten. Das, was Auctex bietet wird man nach und nach entdecken und schätzen lernen.

5.4.1 Installation

Emacs wird durch Emacs-Lisp-Programme erweitert. Die Dateien haben normalerweise die Endung „.el“ . Auch zu Emacs-Lisp hält die Emacs-Homepage viele Informationen bereit. Auf der Auctex-Homepage gibt es den Quellcode. Mit

```
./configure
make
make install
```

in dem Verzeichnis, in das man den Quellcode entpackt hat, kompiliert man Auctex. Hier sollte man bei `./configure` per Optionen Informationen zu seiner Emacsinstallation geben (siehe auch die Installationshilfe auf der Auctex-Homepage[3]). Nun muss Auctex nur noch aktiveriert werden. Dazu trägt man Folgendes in die Konfigurationsdatei ein:

```
(require 'tex-site)
```

5.4.2 Bedienung

Auctex vereinfacht die Bearbeitung von LaTeX-Dokumenten beispielsweise durch Syntaxhighlighting und Kurzbefehle. Das Syntaxhighlighting wird über das Menü

```
Options > Syntax Highlighting
```

[3] siehe Tabelle 5.2 auf Seite 88

aktiviert. Weiteres eher auctexspezifisches findet man unter dem neuen Menü `LaTeX`, in Klammern sind hier oft die jeweiligen Kurzbefehle angegeben. Einen neuen Unterabschnitt fügt man mit beispielsweise mit

```
LaTeX > Section > subsection
```

oder per `Ctrl-c Ctrl-s`. In letzterem Fall wird man von Emacs aufgefordert, weitere Angaben zu machen, konkret (1) was man einfügen will (in unserem Fall `subsection`), welches die default Einstellung bei Artikeln ist, (2) den Titel und (3) das Label. Um eine Umgebung einzufügen gibt man `Ctrl-c Ctrl-e` ein und wird wieder aufgefordert, weitere Details zu nennen. Kleiner Tipp: Mit der Leertaste bekommt man in einem neuen Fenster mögliche Eingaben angezeigt. Auf diese Weise läßt sich beispielsweise eine Aufzählung einfügen.

5.5 Emacs und Preview

Durch Preview ist es möglich, zum einen das Ergebnis von mathematischen Formeln und zum anderen Abbildungen direkt in Emacs gezeigt zu bekommen, quasi WYSIWYG.

5.5.1 Installation

Die Installation ist auf der Auctex Homepage beschrieben.

5.6 Eclipse

Eclipse ist eine Programmierumgebung, das heißt eine Anwendung, die den Programmierer bei seiner Arbeit unterstützen soll. Vielleicht ist das nur so ein Gefühl, das daraus entsteht, dass Eclipse in Java programmiert ist, aber Eclipse eignet sich besonders gut für die Programmierung in Java. Trotzdem: Ähnlich wie Emacs ist auch Eclipse in flexibler Weise erweiterbar, so läßt sich Eclipse für die Programmierung in anderen Programmiersprachen, aber auch noch für ganz andere Dinge anpassen. Als Beispiele kann man hier die Erstellung von Internetseiten und die Erstellung von LaTeX-Dokumenten nennen.

5.6.1 Installation

Für die Installation von Eclipse gibt es mehrere Möglichkeiten. Erstens stehen vorkompilierte Programmpakete zu Verfügung, zweitens kann man sich Eclipse über sein betriebssystemspezifisches Updatesystem nachinstallieren und drittens kann man sich den Quellcode laden und Eclipse selbst kompilieren. Voraussetzung für alle drei Fälle ist, dass Java installiert ist. Im ersten Fall ist die Sache relativ einfach: Man lädt sich das Paket und startet die Installationsroutine. Die Details der Installation können auf der Eclipseseite nachgelesen werden.

5.6.2 Bedienung

Beim ersten Start von Eclipse erscheint zunächst das große Hilfe- und Tutorialfenster. Hier kann man sich das Neuste, aber auch das Altbewährte erklären lassen. Dieses Fenster kann jederzeit über das Menü

```
Help > Welcome
```

erreicht werden. Sonst ist das Hauptfenster in mehrere kleinere Fenster unterteilt. Das wichtigste Fenster ist sicher das Editorfenster. Der Text wird automatisch nach bestimmten Regeln farbig dargestellt. Ein anderes hilfreiches Fenster ist das Navigator-Fenster. Hier sieht man alle Projekte, die im eigenen Arbeitsverzeichnis liegen. Da Eclipse manche Befehle auf alle geöffneten Projekte anwendet, gibt es die Möglichkeit, Projekte zu schließen. So kann man übersichtlich an mehreren Projekten arbeiten und hat immer alle Dateien eines Projektes zusammen. Die Fenster können beliebig angeordnet, versteckt oder sichtbar gemacht werden; vorgefertigte Anordnungen können über das Menü

```
Window > Open Perspective
```

gewählt werden.

Eclipse läßt sich mit Maus und Tastatur steuern. Neben dem Menü, dass in der Standardeinstellung am oberen Bildschirmrand steht, gibt es unterhalb des Menüs noch eine Iconleiste. Die Iconleiste bietet Shortcuts zu Funktionen, die auch über das Menü zu erreichen sind. Oft bietet der berühmte Klick mit der rechten Maustaste zusätzlich ein umfangreiches Menü spezifisch für das markierte Objekt.

Kurzbefehle

Jedes Menü ist über einen Kurzbefehl zu erreichen. Dazu drückt man die `Alt`-Taste zusammen mit dem jeweils unterstrichenen Buchstaben. `Alt-f` öffnet zum Beispiel das File-Menü, mit der Taste `s` kann man dann die Datei speichern, die gerade im Editor geöffnet ist. Hat man mehrere Dateien im Editor, so kann man mit `Alt`-Pfeiltaste zwischen den Editorfenstern springen.

5.6.3 Konfiguration

Eclipse wird über das Menü

```
Window > Preferences
```

konfiguriert. Man erkennt auf den ersten Blick, dass Eclipse eine Softwareentwicklungsumgebung ist. Unter `Window > Preferences > Workbench` läßt sich das Erscheinungsbild von Eclipse anpassen.

5.7 Eclipse und TEXlipse

5.7.1 Installation

Wie bei allen Eclipse-Erweiterungen gibt es zwei Möglichkeiten TEXlipse zu installieren. Erstens kann man sich TEXlipse von der Projektpage herunterladen und einfach in das plug-in-Verzeichnis im Eclipse-Verzeichnis entpacken. Zweitens kann man das Eclipse-Software-Update um die TEXlipse-Update-Seite bereichern, TEXlipse wird dann beim nächsten Software-Update installiert.

5.7.2 Bedienung

TEXlipse bringt auch einen Tabelleneditor mit. Dieser wird über das Menü

```
Window > Show View > Other > Texlipse > Latex Table View
```

sichtbar gemacht. Möglichkeiten offenbaren sich mit einem Klick auf die rechte Maustaste, während der Cursor über dem Tabelleneditor steht.

Kurzbefehle

Mit `Ctrl-s` wird das aktive Dokument gespeichert, mit `Ctrl-b` kann man dann die LaTeX-Bearbeitung für das Projekt anstoßen. `Ctrl-4` zeigt das Dokument, wozu der Viewer benutzt wird, der unter Viewer-Settings eingestellt ist. Mit `Ctrl-Leertaste` läßt man den Codeassistenden zu Wort kommen. Man kann aus einer Liste eine vordefinierte Umgebung auswählen, die dann an der Stelle der aktuellen Cursorposition eingefügt wird. So läßt sich beispielsweise eine eigene Umgebung wie folgt erzeugen:

1. Ctrl-Leertaste
2. Returntaste drücken
3. `environment` markieren und der Umgebung einen eigenen Namen geben

Man vergleiche die Tipparbeit, die man ohne Codeassistenten hätte:

```
\begin{eigenerName}
...
\end{eigenerName}
```

5.7.3 Konfiguration

TeXlipse wird über die Preferences, die man über das Menü

`Window > Preferences` (und dann TeXlipse auswählen)

konfiguriert.

Erzeugen eigener Templates

Bei den vorhandenen Templates fällt auf, dass es insgesamt eher wenige sind. Es sind sozusagen nur die Nötigsten da und sie sind vielleicht auch eher als Beispiele gedacht, denn man kann sich seine eigenen Templates anlegen. Als Beispiel nehmen wir uns nochmal die eigene Umgebung. An die Templates kommt man über das Menü

```
Window > Preferences > Texlispse > Editor > Templates
```

Hier kann man nun das entsprechende Template auswählen. Der Code für die eigene Umgebung sich wie folgt aus:

```
\begin{${environment}}
  ${cursor}
\end{${environment}}
```

Dabei bezeichnet ${Name} eine Variable und ${cursor} ist eine vordefinierte Variable und bezeichnet die Stelle, an die der Cursor gesetzt werden sollen. Nach einem Mausklick auf Edit... erscheint ein Editor. Hier kann man über den Button Insert Variable... weitere vordefinierte Variablen einfügen.

5.7.4 TEXlipse und Aspell

Wie eigentlich alle Programme unter unixartigen Betriebssystemen, so kann auch Eclipse Texte – und unter Unix ist alles eine Textdatei – mit Hilfe von Aspell einer Rechtschreibprüfung unterziehen. Mit `Shift-Ctrl-6` wird die Rechtsschreibprüfung gestartet. Alle Wörter, die nicht korrekt geschrieben oder nicht im Aspellwörterbuch enthalten sind, werden in der Standardeinstellung gelb unterstrichen. Für ein dann markiertes Wort erhält man mit `Ctrl-Leertaste` Korrekturvorschläge. Unter

```
Window > Prefences > Texlipse > Spell Checker
```

muss man Eclipse den Pfad zu Aspell bekannt geben. Unter Apell Arguments kann man Aspell übergeben, in welcher Sprache der Text verfasst ist. LaTeXbefehle werden von Aspell ignoriert.

5.8 Eclipse und CVS

Arbeitet man mit mehreren Personen an einem größeren Projekt, so bietet es sich an, das Projekt zentral zu verwalten. Eclipse unterstützt Concurrent Version System (CVS), ein Open-Source Content Management System. Steht ein CVS Repository zur Verfügung, so kann man über das Menü

```
Window > Show View > CVS Repositories
```

in eine Ansicht mit verschiedenen, CVS betreffenden Fenstern, gelangen. Im linken Fenster kann man dann per Icon CVS+ das Repository eintragen. Für Details zu CVS kann man in der Eclipse-Hilfe oder auf der CVS-Homepage[4] lesen. Mit CVS ist es möglich, dass alle Zugriff auf alle aktuellen Dateien des Projekts haben, Veränderungen verfolgt werden können und ältere Versionen im Fall der Fälle wiederhergestellt werden können.

5.9 Spezielles mit LaTeX

Natürlich kann man mit LaTeX nicht nur gewöhnliche Texte setzen. Die gute Unterstützung für das Setzen mathematischer Formeln ist an anderer Stelle umfassend dargestellt. Will man Musiknoten in Texten (aber auch ohne Text) setzen, kann LaTeX mit Hilfe des Pakets MusixTeX für sich arbeiten lassen. Die Möglichkeiten sind vielfältig, auch wenn die Einarbeitung etwas Zeit in Anspruch nehmen dürfte.

5.10 Weiterführende Informationen

Um an weiterführende Informationen zu kommen kann nur gesagt werden: „Google ist Dein Freund...“. In Tabelle 5.2 schonmal ein paar Treffer.

Linuxe	http://www.Linuxiso.org
Aspell	http://aspell.sourceforge.net/
Aspell (Windows)	http://aspell.net/win32/
Emacs	http://www.gnu.org/software/emacs/
Auctex	http://www.gnu.org/software/auctex/
Preview	http://www.gnu.org/software/auctex/preview-latex.html/
Eclipse	http://www.eclipse.org/
Texlipse	http://texlipse.sourceforge.net/
CVS	http://www.cvshome.org/

Tabelle 5.2: Internet-Links

[4] siehe Tabelle 5.2 auf Seite 88

Kapitel 6

Mathematische Formeln

Hanno Scharfe, Christian Hinkelmann

6.1 Einleitung

Für das Setzen von mathematischen Formeln in LaTeX gibt es mehrere Gründe. Zum einen möchte man in mit LaTeX geschriebenen wissenschaftliche Arbeiten auch mathematische Formeln einbinden. Es ist daher natürlich praktisch diese Formeln dann auch gleich in LaTeX zu setzen.

Zum anderen liefert das Setzen in anderen Formeleditoren (von z.B. Word oder OpenOffice) teilweise nicht das gewünschte Ergebnis.

Gegenüber handgeschriebenen Texten ist der Vorteil natürlich die bessere Lesbarkeit.

6.2 Das `amsmath`-Package

Hanno Scharfe

LaTeX kann schon ohne zusätzliche Packages mathematische Formeln darstellen. Das Package `amsmath`[N.N99], das von der American Mathematical Society entwickelt wird, erweitert die Fähigkeiten von LaTeX, mathematische Formeln darzustellen[Dow02] noch

weiter und sollte daher immer verwendet werden. Zusätzlich sollte man auch noch die Packages `amssymb` einbinden, das weitere mathematische Symbole enthält.

```
\usepackage{amsmath,amssymb}
```

6.2.1 Mathematische Umgebungen

Formeln werden immer in einer mathematischen Umgebung gesetzt, die anders als normaler Text formatiert werden. Es gibt die sehr viele dieser Umgebungen für die unterschiedlichen Einsatzzwecke.

6.2.2 Inline-Umgebungen

Zum einen gibt es die Inline-Umgebungen, mit denen man eine Formel idrekt im laufenden Text darstellen kann. Dies bietet sich für kurze Formeln an, die nicht besonders viel Platz in der höhe benötigen. Ausserdem sollte man es auch für einzelne mathematische Zeichen, beispielsweise Variablen- und Funktionsnamen verwenden. Dadurch werden diese exakt so dargestellt, wie sie auch in einer Formel aussehen. Inline-Umgebungen können von `$` und `$` oder von `\(` und `\)` eingeschlossen werden.

Beispiele:

- `$a^2 + b^2 = c^2$` ergibt $a^2 + b^2 = c^2$.
- Wenn man im Text auf einzelne Variablen, wie b verweisen möchte, schreibt man das so: `$b$`.

6.2.3 Abgesetze Formeln

Längere Formeln sollten immer in eigenen Absätzen angezeit werden, weil sie oft nicht in eine Textzeile hineinpassen. Hierbei gibt es eine sehr grosse Auswahl an Umgebungen:

- `\[ ... \]` und `\begin{equation*} ... \end{equation*}` erzeugen eine Formel, die zentriert in einem Absatz angezeigt wird.

- `\begin{equation} ... \end{equation}` nummeriert Formeln zusätzlich noch. Die Nummerierung wird normalerweise am rechten Rand in runden Klammern angezeigt und jede Formel erhält eine eigene Nummer. Es ist aber auch möglich, die Nummerierung anzupassen.
- `\begin{gather} ... \\ ... \end{gather}` ermöglicht mehrere Formeln in einem Block. Die Formeln erhalten dabei jeweils eine eigene Nummer und werden untereinander, unabhängig voneinander zentriert ausgerichtet. Die Formeln werden dabei durch `\\` voneinander abgetrennt.
- `\begin{align} ...&... \\ ...&... \end{align}` funktioniert ähnlich wie `gather`, bietet aber zuätzlich die Möglichkeit, die Formeln aneinander auszurichten. Dafür setzt man an der Stelle, die jeweils untereinander stehen soll (z.B. vor Gleicheitszeichen) ein `&`. Ausserdem können in einer `align`-Umgebung auch mehrere Formelblöcke nebeneinander platziert werden. Dabei springt man mit jedem zweiten `&` in einer Zeile einen Formelblock nach rechts.
- `\begin{split} ...&... \\ ...&... \end{split}` muss in einer mathematischen Umgebung (z.b. equation) verwendet werden. Damit kann man eine Formel, die nicht in eine Zeile passt, in mehrere Zeilen aufteilen. Die Formeln werden wie auch bei `align` an dem `&`-Symbol ausgerichtet.
- `\begin{multiline} ... \\ ... \end{multiline}` ist eine andere Möglichkeit, lange Formeln in mehrere Zeilen aufzuteilen. Die einzelnen Zielen werden dabei automatisch von links oben nach rechts unten ausgerichtet.

Beispiele:

- Eine Gleichung in einer `equation`-Umgebung:

$$\sum_{n=1}^{\infty} \frac{1}{n(n+1)} = 1 \tag{6.1}$$

- Mehrere ausgerichtete Gleichungen in einer `align`-Umgebung:

$$f_{1,1}(x) = x \qquad f_{2,1}(x) = x \tag{6.2}$$

$$f_{1,2}(x) = x^2 + x \qquad f_{2,2}(x) = \tfrac{1}{2}x^2 + x \tag{6.3}$$

$$f_{1,3}(x) = x^3 + x^2 + x \qquad f_{2,3}(x) = \tfrac{1}{3}x^3 + \tfrac{1}{2}x^2 + x \tag{6.4}$$

6.2.4 Formelnummerierung anpassen

Normalerweise werden Formeln für jedes Kapitel oder bei Dokumentklassen ohne mehrere Kapitel für das ganze Dokument aufsteigend (1), (1), ... nummeriert. Durch den Befehl `\numberwithin{equation}{section}` in der Präambel wird eine weitere Ebene für die Nummerierung eingeführt, wobei die letzte Zahl in jeder neuen Sektion wieder zurückgesetzt wird. Das Egebnis bei Dokumentenklassen ohne mehere Kapitel ist dann (1.1), (1.2), ..., (2.1), (2.2), ...

Mit dem Befehl `\tag{...}` ist es möglich, einer Formel einen eigenen Namen zu geben, was bei wichtigen Formeln, auf die später oft verwirden wird, häufig genuzt wird. Mit `\notag` ist es hingegen mo"glich, für einzelne Formeln in einer Umgebung, die Formeln normalerweise nummeriert (z.B. `align`), die Nummerierung abzuschalten.

Beispiel:

In der `align`-Umgebung, beginnt die erste Zeile mit `\tag{GL}`, die zweite Zeile beginnt mit `\notag` und die dritte Zeile erhält die normale Nummerierung:

$$\mathfrak{f}(\mathfrak{x}) = \mathfrak{t} + \mathfrak{a} + \mathfrak{g} \tag{GL}$$

$$\mathfrak{g}(\mathfrak{x}) = \mathfrak{n} + \mathfrak{o} + \mathfrak{t} + \mathfrak{a} + \mathfrak{g}$$

$$\mathfrak{h}(\mathfrak{x}) = \mathfrak{n} + \mathfrak{o} + \mathfrak{r} + \mathfrak{m} + \mathfrak{a} + \mathfrak{l} \tag{6.5}$$

6.2.5 Verweise auf Formeln

Oft ist es notwenidig, in Text oder in anderen Formeln auf eine Formel verweisen zu können. Diese Verweise werden von LaTeX dann beim Kompilieren automatisch durch die Formelnummer ersetzt. Um auf eine Formel verweisen zu können, muss diese erstmal einen eindeutigen Namen erhalten. Das geht mit dem Befehl `\label{formelname}`, der direkt am Beginn einer Mathumgebung oder am Beginn einer Zeile bei Umgebungen mit mehreren Formeln stehen sollte. Um dann einen Verweis zu erstellen, verwendet man den Befehl `\eqref{formelname}`.

Beispiel:

$$A(r) = \pi * r^2 \tag{6.6}$$

Mit Formel (6.6) lässt sich aus dem Radius eines Kreises sein Flächeninhalt ausrechnen.

6.3 Komplexe Formeln

Christian Hinkelmenn

6.3.1 Setzen von Formeln

Beim Setzen von Formeln muss man einige Besonderheiten des Mathemodus beachten. Durch den Mathemodus werden alle Buchstaben und Lehrzeichen wie Variablen behandelt, d.h. Buchstaben werden kursiv gestellt und die Lehrzeichen werden entfernt.

1.Beispiel:

Aus `A = \{a + b mit a = 3 und b = 5 \}` wird

$$A = \{a + bmita = 3undb = 5\}$$

Um das kursiv Setzen zu verhindern, benutzt man den Befehl `\text`.

2.Beispiel:

Aus `A = \{a + b\text{ mit }a = 3\text{ und }b = 5 \}` wird

$$A = \{a + b \text{ mit } a = 3 \text{ und } b = 5\}$$

6.3.2 Mathematische Konstrukte

Des weiteren gibt es in Mathematischen Formeln natürtlich auch Konstrukte, die nicht einfach nur nebeneinander geschrieben werden. Als erstes wären da die Brüche. Sie werden mit dem Befehl `\frac{1}{6}` gesetzt, wobei der erste Parameter der Zähler ist und der zweite Parameter ist der Nenner des Bruches.

Anzeigebesispiel:

$$\frac{1}{6}$$

Wurzeln werden mit dem Befehl `\sqrt[4]{a}` gesetzt. Hierbei ist der erste Parameter optional und gibt an die wievielte Wurzel dargestellt werden soll. Wird dieser Parameter nicht angegeben so wird an der Stelle nichts angezeigt. Der zweite Parameter ist der Radikant (der Wert unter der Wurzel).

Anzeigebesispiel:

$$\sqrt[4]{a}$$

Potenzen werden mit dem Zeichen ^ gesetzt. Falls mehr als ein Zeichen hochgestellt werden soll, muss der hochzustellende Teil in geschweifte Klammern gesetzt werden (`a^{b+c}`).

Anzeigebeispiel:

$$a^{b+c}$$

Eine Summe wird mit dem Befehl `\sum` gesetzt. Nach der Summe kann man dann weiterschreiben für die Summanden (`\sum \frac{1}{6}`).

Anzeigebeispiel:

$$\sum \frac{1}{6}$$

Wenn noch Indizes an der Summe stehen sollen, so werden diese mit dem Befehl für Tiefstellen bzw. fürs Hochstellen gesetzt

(`\sum_{i=1}^{\infty} \frac{1}{i}`).

Hierbei gilt es allerdings zu beachten, dass diese „Parameter“ nur bei abgesetzten Formeln über bzw. unter dem Summenzeichen dargestellt werden. Bei Inline-Formeln werden die Angaben ganz normal Hoch- bzw. Tiefgestellt angezeigt (**Beispiel (Inline)** $\sum_{i=1}^{\infty} \frac{1}{i}$).

Anzeigebeispiel (abgesetzte Formel):

$$\sum_{i=1}^{\infty} \frac{1}{i}$$

Integrale werden wie die Summe gesetzt, allerdings mit dem Befehl `\int` anstatt dem Befehl `\sum` (`\int_{1}^{2}x\text{ dx}`).

Anzeigebeispiel (abgesetzte Formel):

$$\int_1^2 x \, dx$$

6.3.3 Griechische Buchstaben

Griechische Buchstaben werden mit dem ihnen entsprechenden Befehl gesetzt. Der Befehl besteht aus einem Backslash und dem Namen des Buchstaben. Dabei werden kleine Buchstaben z.B. γ kleingeschrieben (`\gamma`) und die Großen z.B. Γ werden Großgeschrieben (`\Gamma`).

Außerdem gibt es dann noch die „Var“-Version von einigen kleinen Buchstaben. Diese haben zwischen dem Backslash und dem Namen noch das Wort „var“ stehen. (z.B. `\varepsilon`). Es gibt „Var“-Version von Epsiolon ($\epsilon \to \varepsilon$), Phi ($\phi \to \varphi$), Pi ($\pi \to \varpi$), Rho ($\rho \to \rho$), Sigma ($\sigma \to \sigma$) und Theta ($\theta \to \vartheta$)

6.3.4 Klammern

Es gibt drei verschiedene Klammerungsarten für mathematische Formeln. Text Klammern (werden z.B. durch `[...]` gesetzt) diese haben immer die gleiche Größe, die zwar für Text funktioniert. Aber bei mathematischen Formeln (die ja in der Höhe von Text abweichen können) sehen die Klammen nicht gut aus.

Anzeigebeispiel:

$$[\sum_i a_i \left| \sum_j x_{ij} \right|^p]^{1/p}$$

Bei automatische Klammern (werden z.B. durch `\left[...\right]` gesetzt), wird die Größe an die Formel angepasst. Allerdings sieht das auch nicht immer gut aus, wie das folgende Beispiel zeigt.

Anzeigebeispiel:

$$\left[\sum_i a_i \left| \sum_j x_{ij} \right|^p \right]^{1/p}$$

Bei dem vorrigen Beispiel passen festgelegte Klammern (werden z.B. durch `\biggl[...\biggr]]` gesetzt) besser. Diese haben aber den Nachteil, dass man die Größe direkt festlegen muss und daher wissen muss wie hoch sie sein müssen.

Anzeigebeispiel:

$$\left[\sum_i a_i \left|\sum_j x_{ij}\right|^p\right]^{1/p}$$

6.3.5 Punkte

Im Mathemodus hat man die Auswahl aus verschiedenen (Auslassungs-) Punkten, die verschiedene Anwendungen haben.

Innerhalb von Aufzählungen benutzt man den Befehl `\dotsc`.

Anzeigebeispiel:

$$a_1, \ldots, a_n$$

Bei Auslassungen innerhalb von Operationen benutzt man den Befehl `\dotsb`.

Anzeigebeispiel:

$$a_1 + \cdots + a_n$$

Bei Auslassungen von Faktoren bei Multiplikationen benutzt man den Befehl `\dotsm`.

Anzeigebeispiel (Um den Unterschied zu zeigen das · durch * ersetzt):

$$a_1 * \cdots * a_n$$

Bei Auslassungen innerhalb von Integralen benutzt man den Befehl `\dotsi`.

Anzeigebeispiel:

$$\int_{A_1}\int_{A_2} \ldots$$

Außerdem gibts dann noch die vertikalen (`\vdotso`) und diagonalen (`\ddotso`) Auslassungspunkte.

6.3.6 Schriftarten

Es gibt die drei Schriftarten mathcal,mathbb und mathfrac. Die ersten beiden unterstützen keine Kleinbuchstaben. Die Schriftart wird umgeschaltet, indem erst der Backslash dann der kleingeschriebene Name der Schriftart und zuletzt als Parameter dahinter der zu setzende Text folgt.

Anzeigebeispiel mathcal:

$$\mathcal{ABCDEFGHIJKLMNOPQRSTUVWXYZ}$$

Anzeigebeispiel mathbb:

$$\mathbb{ABCDEFGHIJKLMNOPQRSTUVWXYZ}$$

Anzeigebeispiel mathfrak:

$$\mathfrak{ABCDEFGHIJKLMNOPQRSTUVWXYZ}$$

$$\mathfrak{abcdefghijklmnopqrstuvwxyz}$$

6.3.7 Funktionen

Bei Funktionen besteht das selbe Problem wie beim schreiben von Text in einer Matheumgebung. Die einzelnen Buchstaben werden kursiv gesetzt. Um dies zu verhindern gibt es vordefinierte Funktionen wie z.B. Sinus (`\sin`) und Cosinus (`\cos`).

Anzeigebeispiel:

$$\sin^2(x) + \cos^2(x) = 1$$
$$\sin^2(x) + \cos^2(x) = 1$$

Zusätzlich zu den diversen vordefinierten Funktionen gibt es die Möglichkeit eigene Funktionen zu definieren. Die Befehle hierfür lauten `\DeclareMathOperator` und `\providecommand`. Der erste Befehl erstellt eine Funktion, die späeter keine Parameter bekommt. Hinter dem Befehl `\DeclareMathOperator` folgen zwei Parameter. Der erste Parameter gibt den Befehl an, der die Funktion aufruft. Der zweite Parameter gibt die Ausgabe der Funktion an.

Der Befehl `\providecommand` bekommt zuerst einen Pflichtparameter, der den Befehl angibt mit dem die Funktion aufgerufen wird. Dann folgt ein optionaler Parameter, der aus einer Zahl besteht und angibt wieviele Parameter die Funktion später übergeben bekommen soll. Als letzter Parameter folgt dann wieder ein Pflichtparameter, der Angibt wie die Ausgabe aussehen soll. Hierbei werden dann die eventuell vorhandenen Parameter der neuen Funktion eingebunden. Per `#1` bekommt man z.B. den Inhalt des 1. Parameters.

Die beiden Befehle sollten in der Präambel stehen.

So siehts in LaTeX aus:

```
...
\DeclareMathOperator{\esssup}{ess\,sup}
\providecommand{\abs}[1]{\lvert#1\rvert}

\begin{document}
\abs{z}
\esssup(y,z)
...
```

Anzeigebeispiel:

$$|z|$$
$$\operatorname{ess\,sup}(y, z)$$

Es gibt vier verschiednene Befehle für Modulo-Funktionen.

Der erste Befehl (`\mod`) erstellt vor dem mod einen relativ großen Abstand. $a \mod b$

Der zweite Befehl (`\bmod`) erstellt fast die selbe Ausgabe wie der erste Befehl, allerdings ohne den größeren Abstand. $a \bmod b$

Der dritte Befehl (`\pmod`) erstellt fast die selbe Ausgabe wie der zweite Befehl, nur dass das `mod b` in Klammern gesetzt wird. $a \pmod b$

Der vierte und letzte Befehl (`\pod`) erstellt fast die selbe Ausgabe wie der dritte Befehl, aber ohne das `mod`. $a \pod b$

6.3.8 Notationen

In Formeln werden des öfteren auch Umformungen durchgeführt oder Folgerungen gezogen. Hierzu benötigt man verschieden Arten von Pfeilen. LaTeX bietet dazu verschiedene Befehle, die aber ähnlich gebildet werden. Nach einem Backslash folgt die Richtung des Pfeils (z.B. „right“) und dann „arrow“. Der entstande Befehl (`\rightarrow`) liefert dann einen einfachen, nach rechts zeigenden Pfeil ($\rightarrow$). Zusätzlich zu „left“ und „right“ gibt es

noch die Richtung „leftright“. Dies liefert einen Pfeil mit einer Pfeilspitze auf der linken Seite und einer auf der rechten Seite ($\leftrightarrow$).

Es gibt die Möglichkeit den Pfeil zu verlängern. Hierzu muss nur vor der Richtung des Pfeils ein „long“ stehen. Dadurch wird aus einem kurzen Pfeil ($\leftrightarrow$) ein langer Pfeil ($\longleftrightarrow$).

Soll der Pfeil nun eine doppelte Linie haben (wie bei mathematischen Umformungen üblich), so muss man den ersten Buchstaben des Befehls groß schreiben (`\Leftrightarrow`).

Außerdem kann man einen Pfeil noch beschriften. Dies wird mit Hilfe des `\stackrel{(6.2)}{\Longleftrightarr` getan. Dieser Befehl dient dazu 2 Konstrukte übereinander zu schreiben.

Anzeigebeispiel:

$$\stackrel{(6.2)}{\Longleftrightarrow}$$

Um Zusammenfassungen in Formeln zu verdeutlichen gibt es die Befehle `\underbrace{}_{}` (Unterklammerung) und `\overbrace{}^{}` (Überklammerung). Der erste Parameter gibt an, was über bzw. unterklammert werden soll und der zweite Parameter gibt an was an dieser Klammer stehen soll. Der Unterstrich bzw. das ^ müssen dort stehen.

Anzeigebeispiel:

$$\underbrace{10 + 5}_{15} + 20 = \overbrace{15 + 20}^{35} = 35$$

Zugehöhriger LaTeX-Code:

```
\underbrace{10 + 5}_{15} + 20 = \overbrace{15 + 20}^{35} = 35
```

6.3.9 Vektoren und Matrizen

LaTeX macht keinen Unterschied zwischen Vektoren und Matrizen. Beide werden mit dem selben Befehl gesetzt. Hierbei wird ein Vektor als eine einspaltige Matrix angesehen. Für eine Matrix wird eine eigene Umgebung geo"ffnet (`\begin{pmatrix} ... \end{pmatrix}`).

Um die Zeile innerhalb einer Matrix zu wechseln benutzt man einen doppelten Backslash (\\) und um die Spalte zu wechseln das Zeichen `&`. Dabei wird erst eine Zeile zu ende geschrieben, bevor man die nächste anfängt.

Um bei einem Vektor v den Vektorpfeil über das v zu schreiben, wird der Befehl `\vec{v}` benutzt. Der Parameter gibt dabei den Namen des Vektors an, der unter dem Pfeil steht

Anzeigebeispiel Vektor:

$$\vec{v} = \begin{pmatrix} 5 \\ 2 \\ 3 \\ 9 \end{pmatrix}$$

Anzeigebeispiel Matrix:

$$M = \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix}$$

So sieht die Matrix in LaTeX aus:

```
M = \begin{pmatrix} 1 & 2\\ 3 & 4\\ 5 & 6 \end{pmatrix}
```

6.3.10 Abstände

Im Mathemodus werden Leerzeichen wie bereits erwähnt nicht beachtet. Es kann allerdings mit dem `\text` Befehl trotzdem ein (oder mehrere) Leerzeichen als Abstand gesetzt werden. Soll aber nun der Abstand kleiner oder größer sein, als ein Leerzeichen so braucht man besondere Befehle zum setzen.

Es gibt fünf verschiedene Abstandsgrößen im Mathemodus. Den kleinen Abstand (`A\,B`), den mittleren Abstand (`A\:B`), den großen Abstand (`A\;B`), sowie den quad-Abstand (`A\quad{}B`) und den qquad-Abstand (`A\qquad{}B`)

Anzeigebeispiel:

Kein Abstand: AB

Kleiner Abstand (\thinspace): $A\,B$

Mittlerer Abstand (\medspace): $A\:B$

Großer Abstand (\thickspace): $A\;B$

Abstand mit \quad: $A\quad B$

Abstand mit \qquad: $A\qquad B$

6.3.11 Displaystyle-Befehl

Der Befehl `\displaystyle` erzwingt das Setzen eines Ausdruckes in der richtigen Größe. Ohne den Befehl werden (wie im folgenden Beispiel) bei Brüchen die Indizes von Summen- und Produktzeichen falsch gesetzt.

Anzeigebeispiel (ohne den Befehl):

$$\frac{\sum_{n>0} z^n}{\prod_{1\leq k\leq n}(1-q^k)}$$

Der Befehl `\displaystyle` wird innerhalb einer extra geöffneten Umgebung benutzt. Er sorgt in diesem Beispiel dafür, dass die Indizes richtig gesetzt werden. Außerdem wird die Größe des Nenners bzw. des Zählers nicht mehr angepasst, wie man an den Produkt- bzw. Summenzeichen im Beispiel erkennen kann.

Anzeigebeispiel (mit dem Befehl):

$$\frac{\displaystyle\sum_{n>0} z^n}{\displaystyle\prod_{1\leq k\leq n}(1-q^k)}$$

6.4 Theoreme, Definitionen und Beweise

Hanno Scharfe

In vielen Dokumenten mit mathematischen Formeln werden diese zu Theoremen oder ähnlichem zusammengefasst. Auch hierfür gibt es in LaTeX spezielle Umgebungen, die das Aussehen der Theoreme vereinheitlichen. Eine einfache Version davon ist bereits in LaTeX integriert, es ist aber empfehlenswert, das Package `amsthm`[N.N04] zu verwenden, was unter anderem weitere Formatierungen anbietet. Dies geht mit dem Befehl `\usepackage{amsthm}`.

6.4.1 Theoremumgebungen definieren

Nun können in der Preambel des Dokuments belibige Theoremumgebungen definiert werden. Dafür verwendet man dann den Befehl `\newtheorem{name}{text}`. Wenn sich

verscheidene Typen von Theoremen die gleiche Nummerierung teilen sollen (z.B: Definition 1, Lemma 2, Definition 3, ...), kann man dies in einem optionalen Parameter angeben. Durch `\newtheorem{lem}[defi]{Lemma}` wird, wenn vorher eine Umgebung mit dem Namen `definition` definiert wurde, der Umgebung `lemma` die gleiche Nummerierung zugewiesen.

6.4.2 Theoremumgebungen verwenden

Um die vorher definiderten Theoremumgebungen zu verwenden, wird einfach der Name dieser Umgebung in einem `\begin{name} ... \ end{name}` Block verwendet. Diesem kann in einem optionalen Parameter noch einen zusätzlichen Namen erhalten: `\begin{satz}[Ein Satz] ... \end{satz}`. Ausser den selbst definiderten Theoremumgebungen existiert immer auch eine vordefiniderte Beweisumgebung. Diese wird mit `\begin{proof}Hier ein Beweis... \end{proof}` verwendet. In einer Beweisumgebung wird automatisch das qed-Symbol □ an das Ende gesetzt. Wenn es nicht am ende stehen soll, weil man hinter dem Abschluss des Beweises noch zusätzlichen Text schreiben möchte, kann man in der Zeile, in der das Symbol erscheinen soll, den befehl `\qedhere` verwenden.

6.4.3 Anpassen der Darstellung

Um das Aussehen der unterschiedlichen Theoremumgebungen anzupassen, ist es möglich, eigene Styles dafür zu erstellen. Man kann aber auch zwischen den vordefiniderten Styles `plain`, `definition` und `remark` wählen, die normalerweise ausreichend sind. Wie diese vordefinierten Styles dargestellt werden, hängt von der Dokumentenklasse ab. Um einen Styles für eine Theoremumgebung festzulegen, wird vor der Definition dieser Umgebung der Befehl `\theoremstyle{style}` verwendet. Voreingestellt ist dabei das Design `plain`. Ausserdem ist es möglich, die Nummerierung bei einzelnen Theoremen abzuschalten, die Nummerierung links oder rechts auszurichten und eine abschnittsweise Nummerierung zu verwenden. Wie das funktioniert, steht in [N.N04].

Beispiele:

In der Preambel:

```
\usepackage{amsthm}
\theoremstyle{definition}
\newtheorem{defi}{Definition}
\theoremstyle{plain}
\newtheorem{lem}[defi]{Lemma}
\newtheorem{satz}{Satz}
```

In dem Dokument:

```
\begin{satz}[Der Satz von I. Rgendwem]
Hier wird ein Satz gesetzt...
\end{satz}
\begin{defi}
Dazu sollte nat"urlich etwas definiert werden...
\end{defi}
\begin{lem}
Wir brauchen mal kurz ein Lemma...
\end{lem}
\begin{proof}
Beweis des Lemmas...
\end{proof}
```

Das Ergebnis:

Satz 1 (Der Satz von I. Rgendwem)**.** *Hier wird ein Satz gesetzt...*

Definition 1. Dazu sollte natürlich etwas definiert werden...

Lemma 2. *Wir brauchen mal kurz ein Lemma...*

Beweis. Beweis des Lemmas... □

6.5 Einheiten

Christian Hinkelmann

Wenn man z.B. bei einem physikalischen Text immer den gleichen Abstand zwischen Einheit und Zahlenwert haben möchte, so kann man dazu das Package Units verwenden. Des Weiteren verhindert der Befehl automatisch das „kursiv stellen“ der Buchstaben (bei der Einheit). Mit `\usepackage{units}` kann man das Package einbinden.

Das Package bietet den Befehl `\unit`, gefolgt von einem optionalen Parameter (dem Wert vor der Einheit) und einem Pflichtparameter (der Einheit selber). Ein vollständiger Befehl wäre dann `\unit[17]{V}`. Dieser Befehl muss in einer Mathematischen Umgebung benutzt werden.

Beispiel:

```
An dem Stromkreis liegt eine Spannung von $\unit[220]{V}$ an.
In dem Stromkreis befindet sich ein Widerstand mit 10 $k\Omega$
und ein weiterer Widerstand mit $\unit[1]{k\Omega}$.
```

Das Ergebnis:

An dem Stromkreis liegt eine Spannung von 220 V an. In dem Stromkreis befindet sich ein Widerstand mit 10 kΩ und ein weiterer Widerstand mit 1 kΩ.

Am Beispiel kann man gut erkennen, dass bei den 10 kΩ der Abstand länger ist als bei dem Unitsbefehl und das k kursiv gesetzt wurde.

6.6 Graphen zeichnen

Hanno Scharfe

Eine weitere wichtige Funktion in mathematischen Doukmenten ist das Zeichnen von Graphen. Auch dies ist mit LaTeX möglich. Relativ einfach kann man das mit dem Package `pst-plot` [Voß02] (`\usepackage{pst-plot}`) aus dem Paket PSTricks (http://tug.org/PSTricks) erreichen. Dieses verwendet allerdings Funktionen, die nur bei PostScript zur Verfügung stehen. Um ein pdf-Dokument zu erzeugen, ist es daher nötig, zuerst eine PostScript-Datei zu erzeugen und diese danach in ein pdf-Dokument zu konvertieren. Wenn man nun eine Funktion zeichnen möchte, muss man zuerst eine `pspicture`-Umgebung öffnen. In dieser Umgebung kann man dann verschiedene Zeichenfunktionen verwenden. Beim

öffnen der Umgebung kann man direkt die logischen Koordinaten angeben, die die Ecken links unten und rechts oben haben sollen.

6.6.1 Einige wichtige Befehle

`\psline[linewidth=1pt]{->}(-20,0)(400,0)`
Dieser Befehl erzeugt eine Linie. In dem optionalen Argument kann die Linienstärke angegeben werden. Danach wird der Pfeiltyp angegeben (hier ein Pfeil zum Zielpunkt). Als letztes folgen der Start und der Zielpunkt, die jeweils in logischen Koordinaten angegeben werden.
`\rput(-5,1.75){text}`

Durch diesen Befehl kann Text an eine bestimmte Position geschrieben werden. Der Text wird dabei vertikal Zentriert ausgegeben.
`\psplot[plotstyle=curve,linewidth=0.5pt]{-20}{400}{x sin}`

Dies ist der wichtigste Befehle, durch den der Funktionsgraph gezeichnet wird. Dabei werden funktioswerte für einzelne Punkte berechnet und durch Linien verbunden. `plotstyle=curve` bedeutet dabei, dass die Verbindungslinien Kurven sein sollen, wodurch der Graph etwas besser aussieht. Die nächsten beiden Argumente sind dabei der Start- und der Endwert, der gezeichnet werden soll. Als letztes wird die eigentliche Funktion angegeben (hier `x sin`). Diese Funktion wird immer in der Umgekehrten Polnische Notation angegeben, zuerst kommen also alle Argumente einer Funktion und danach folgt der Name der Funktion selber.

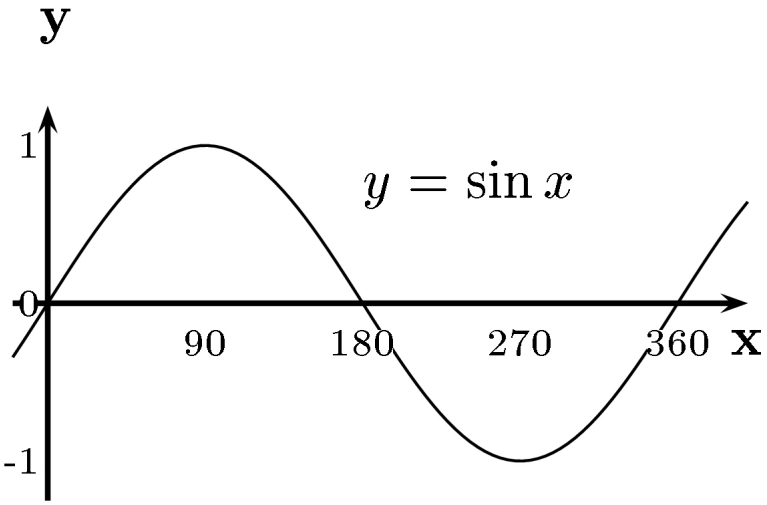
Beispiel:

Durch diese Befehle wird eine Sinuskurve im Bereich -20 bis 400 Grad gezeichnet. Zusätzlich werden die Koordinatenachsen gezeichnet und beschriftet.

```
\begin{pspicture}(-20,-1.25)(400,1.25)
%Achsen zeichnen
\psline[linewidth=1pt]{->}(-20,0)(400,0)
\psline[linewidth=1pt]{->}(0,-1.25)(0,1.25)
%Achsen beschriften
```

```
\multido{\n=+90}{5}{\rput(\n,-0.25){\scriptsize \n}}
\multido{\n=-1+1}{3}{\rput[r](-4,\n){\scriptsize \n}}
\rput[l](-5,1.75){$\mathbf{y}$}
\rput[l](390,-.25){$\mathbf{x}$}
\rput[l](180,0.75){$y=\sin x$}
%Graph zeichnen
\psplot[plotstyle=curve,linewidth=0.5pt]{-20}{400}{x sin}
\end{pspicture}
```

Das Ergebnis:



Kapitel 7

Collaborative Writing

Hier fehlt noch der Beitrag!

Kapitel 8

Internet und neue Medien

Kirsten Albrecht, Natalia Ditz

8.1 Ein internetfähiges Dokument

Welche Eigenschaften sollte ein Dokument haben um internetfähig zu sein? Es sollte über Hyperlinks verfügen. Also es sollte möglich sein von einer Schlüsselstelle in dem Dokument zu einer anderen zu springen. Z.B von einem Unterpunkt in einem Inhaltsverzeichnis in entsprechenden Abschnitt und wieder zurück zum Inhaltsverzeichnis. Es sollte möglich sein in ein externes Dokument oder auf eine Internetseite zu springen. Eventuell könnte man verlangen den HTML-Abfrageformularen ähnliche Formulare und eine Auswertung zu ermöglichen.

8.2 CSS

CSS steht für Cascading Style Sheets. CSS ist eine Formatvorlage für ein HTML-Dokument. Das Konzept von CSS ist der folgende: die Spezifikationen des Darstellungsformats sollten vom eigentlichen Dokument trennen. Das heißt die Form eines HTML-Dokuments sollte besser von seinem Inhalt getrennt werden. Für das Einsetzen von CSS haben sich folgende Vorteile erwiesen:

- es ist eine wesentlich erweiterte Kontrolle des Layouts möglich,

- man kann eigenen Satz an speziellen HTML-Erweiterungen definieren,
- man kann Hintergründe, Schriftarten und Schriftgrößen problemlos spezifizieren und bei Bedarf leicht austauschen,
- Definitionen des Dokumentslayouts werden als getrennte Dateien gespeichert, so mit können mehrere CSS-Dateien in einem HTML-Dokument oder eine CSS-Datei in verschiedenen HTML-Dokumenten eingesetzt werden.

Es gibt auch Nachteile. Ein Nachtteil ist, dass nicht jeder Browser CSS-Dateien unterstützt und es zu unerwünschten Ergebnissen kommen kann. Es kann auch passieren, dass ein Browser, der die CSS-Dateien unterstützt, diese fehlerhaft darstellt.

8.3 von latex zu Html

Sowohl das Internet, wie auch Latex waren bei ihrer Entwicklung dazu gedacht Wissen zu verarbeiten. Ersteres sollte helfen Wissen möglichst vielen Leuten zugänglich zu machen, zweiteres um ein einfaches Handwerkszeug zu haben, um wissenschaftliche Texte, die nicht selten komplexe Ausdrücke enthalten, bequem und gut darzustellen. Es liegt also nahe beide zu verknüpfen, das heißt zunächst einen Text zu verfassen und ihn dann im Internet vielen Leuten zur Verfügung zu stellen. Damit man nicht jedes Document zweimal schreiben muss, wurden Konverter entwickelt, die Latex-Code in Html-Code umwandeln. Zwei dieser Konverter sollen hier vorgestellt werden: latex2html und tex4ht.

8.3.1 latex2html

Diesem auf Pearl basierenden Konverter kann ein fertiges Latex-Dokument übergeben werden, ohne das vorher etwas an ihm geändert werden muss. Bei der Umwandlung von Latex zu html wird alles, was sich nicht einfach mit einem Standardzeichensatz darstellen lässt, durch Bilder dargestellt. Dies gilt insbesondere für die meisten mathematischen Formeln. Auch Bilder aus dem Ursprungsdokument werden in ein anderes Format konvertiert (.png). Mit den Defaulteinstellung eignet sich dieser Konverter besonders für umfangreiche Documente, da er ohne besondere Einstellung für eine große Knotentiefe sorgt, das heißt, dass das Ursprungsdokument, gemäß seinen Kapiteln und Unterkapiteln, in viele kleinere Internetseiten zersplittert wird.

Installation

Bevor man Latex2html selbst installieren kann, muss man zunächst die Vorraussetzungen schaffen: Wie oben erwähnt ist Latex2html in Pearl geschrieben, dass heißt man muss zunächst Pearl installieren (was relativ unkompliziert ist). Nachdem das auf dem Rechner ist, braucht man als nächstes netpbm. Dann läht man latex2html runter und passt die Datei pstoimg an. Dann kann man die config starten. Nun kann man den mitgelierferte test starten (test.bat). Wenn alles gut gegangen ist, dass heißt eine Internetseite erstellt wurde, kann man jetzt 'install' starten und ist fertig, wenn nicht sollte man nochmal prüfen ob alle Angaben in pstoimg korrekt sind oder einfach google fragen, was insgesammt bei dieser Installation nicht schaden kann.

Anwendung

Latex2html erzeugt normalerweise bei der Installation einen neuen Ornder Latex2html in dem man die latex2html.exe findet. diese ruft man in der Konsole auf und schreibt dahinter durch ein Leerzeichen getrennt die Datei (oder Dateien) die man konvertieren möchte. Nun wird ein neuer Ordner in dem gleichem Verzeichnis erzeugt, in dem die ursprünglich Latex-Datei lag. In diesem Ordner müssten sich, nachdem der Konverter fertig ist, alle Dateien befinden, die zu der fertigen Internetseite gehören. Wenn einem die Standarteinstellungen nicht gefallen kann man diese auf untersschiedlichen Wegen beeinflussen. der Wohl einfachste Weg ist beim Aufruf von Latex2html vor den Dateien noch Optionen anzugeben. z.B. kann mit '-split x' (x = 0,...,4) die Knotentiefe verändert werden. Oder aber man ändert den Standartinfotext, den latex2html zu jeder Internetseite erzeugt, mit -info "neuer Infotext". Wie diese beiden Optionen schon andeuten gibt es an dieser Stelle schon vielfältige Methoden das Endprodukt zu beeinflussen. Wenn einem dieser Weg nicht gefällt oder man keine passende Option in den Dokumentationen findet, hat man natürlich noch andere Möglichkeiten. Zum Beispiel kann man den Kode von latx2html verändern. Bei den Originaldateien ist Vorsicht geboten. Wenn man die ändern möchte, muss man Latex2html danach noch mal installieren. Dieser Weg ist gerade für Anfänger wohl er schwer. Aber man kann auch einfach neue Dateien erzeugen und Latex2html 'sagen', dass diese mitbenutzt werden sollen. Folgende Datiei währe zum Beispiel eine Möglichkeit:

#myinit.pl

$MAX_SPLIT_DEPTH = 0; # beeinflusst die Knotentiefe
$NO_NAVIGATION = 1; # schaltet die Navigationszeile aus.

In der Kommandozeile kann man die Datei wie folgt einbinden: 'latex2html.bat -init_file myinit.pl Beispieldatei.tex' Eine weitere Möglichkeit der Manipulation besteht darin den Quelltext des Latex-Dokument leicht zu verändern, diese Änderungen müssen nicht das Latex-Dokument selbst bertreffen. Auch hierzu möchte ich ein paar kleine Beispiel geben:

- Link in die Kommandozeile einfügen:
  \htmladdtonavigation{\htmladdnormallink{neuerLink} {http:\\www.uni-hamburg.de}}
- Die Links zu 'Kindern' ein bzw ausblenden:
  \tableofchildlinks[off|on|all|none]
- keine Informationsseite erstellen:
  \htmlinfo[of]
- Head erweitern:
  \htmlhead[center]{section}{Text}
- Befehle nur für Latex bzw. html
  \begin{latexonly}, \latex, %\begin{latexonly}
  \begin{htmlonly}, \html

8.3.2 tex4ht

Auch diesem Konverter kann man im Prinzip denn fertigen Latex-Code übergeben. Allerdings muss hier die Zeile \usepackage{tex4ht} einfgeügt werden. Vom Prinzip her funktionieren Latex2html und tex4ht sehr ähnlich: Auch tex4ht erstellt für alles, was sich nicht einfach mit einem Standardzeichensatz darstellen lässt Bilder. Einen wesentlichen Unterschied merkt man, wenn man die Defaulteinstellungen benutzt, während latex2html eine große Knotentiefe wählt, erstellt tex4ht nur eine Seite, sorgt also für eine geringe knotentiefe.

Installation

Die Installation von tex4ht ist relativ unkompliziert. Ein wichtiger Hinweis ist das es sehr unterschiedliche Installationsanleitungen im Internet gibt, die leider nicht alle funktionieren. Eine mit der es geklappt hat, ist unter der Adresse http://www.cse.ohio-state.edu/~gurari/TeX4ht/mn-mswin.html zu finden.

Anwendung

Wie schon oben erwähnt muss als erstes die Zeile \usepackage{tex4ht} eingefügt werden. Die .tex Datei, die man in eine Internetseite umwandeln möchte, sollte man am besten an dem Ort ablegen, den man bei der Installation von tex4ht als Arbeitsverzeichnis angegeben hat (z.B. Eigene Dateien). Dann kann man in der Kommandozeile mit 'htlatex documentname.tex' einfach die Kovertierung starten. im gleichen Ordner wie dem aus dem die Texdatei stammt findet man dann das Ergebniss. Wenn einem das Ergebnis nicht gefällt, gibt es unterschiedliche Möglichkeiten das Ergebnis zu ändern. Wie bei vielen Packeten, kann man auch für tex4ht einige Optionen angeben. Vom Aufbau her sieht das wie folgt aus: \usepackage[html,option,...]{tex4ht}. Ein sehr nützliches Beispiel für eine Option ist die Knotentiefe, dazu wählt man einfach eine natürliche Zahl bis 4 und gibt die als Option an. Eine andere nette Option ist das erstellen von Links zu den einzelnen Kapiteln und zurück (über die Überschriften), dazu sagt man einfach 'sections+'. Eine weitere Möglichkeit das Endprodukt zu beeinflussen stellen CSS-Befehle dar, die man im Latex-Quellcode einfügen kann:

- \Css{.maketitle{ border: solid 5px; width: 100\%}} %erzeugt Box
- \Css{.sectionHead {
  text-align: right; %setzt die Überschriften rechts
  font-family: cursive; %Ändert den Überschriftstyp auf Kursive
  border-bottom: solid 2px; }}% zeichnet eine Linie unter die Überschriften
- \Css{body { font-family: cursive; }} %Setzt den Allgemeinen Schrifttyp

Am Rande erwähnt sei noch wie man einen zusätzlichen Link einfügt: \Link[http://www.uni-hamburg.de]{}{}Uni\EndLink. Wenn man direkt im Latex-Code auch Html-Anweisungen geben möchte kann man das mit dem Befehl '\HCode{Inhalt}' machen.

8.3.3 Latex2html vs tex4ht

Beide sind nicht ganz einfach für einen Laien zu installieren, wobei tex4ht noch deutlich einfacher zu installieren ist als Latex2html. tex4ht ist einfacher zu beeinflussen, dafür muss man bei Latex2html nicht den Quelltext beeinflussen. Das Einfügen des Packages tex4ht hat unschöne Seiteneffekte beim Erzeugen eines PDF, solche Effekte gibt es bei latex2html nicht. Es ist sicherlich auch ein positiver Punkt für Latex2html, dass der Konverter einen eigenen Ordner für die Internetdateien erzeugt. Mit beiden Konvertern kann man einfach und schnell eine fertige Internetseite erzeugen, das heißt beide Werkzeuge erfüllen ihren Zweck. Ihr Defaultdesign ist sehr unterschiedlich: Latex2html bevorzugt eine große Knotentiefe, tex4ht eine kleine. Vom Prinzip her, kann man fast alles was der eine machen kann, auch mit dem anderem machen. Wobei tex4ht wesentlich einfacher für unbedarfte zu beeinflussen ist, als Latex2html.

8.4 Hyperrefpackage

8.4.1 Hyperlinksdarstellung

Um ein internetfähiges Dokument mit Latex zu bekommen, braucht man den Hyperrefpackage. Allerdings wirkt sich dieser Package bevorzugt auf PDF-Dokumente aus.

Mit \usepackage {hyperref} in der Präambel wird das Paket im Dokument bekannt gemacht. Es werden mehrere Optionen angeboten, die die Darstellung der Links beeinflussen. Einige davon sind:

- breaklinks
- colorlinks
- backref
- pagebackref

Die Option *breaklinks* erlaubt Zeilenumbruch innerhalb eines Verknüpfungstextes. Leider wird dieses bei dvips nicht unterstützt. Die Option *colorlins* bestimmt die Farbe von

Verknüpfungen und Anker im Dokument Die Option *backref* ermöglicht Rückverknüpfung als Liste von Abschnittsnummern darzustellen. Die Option *pagebackref* stellt die Rückverknüpfungen als Liste von Seitennummern dar.

8.4.2 Benutzermakros für Hyperlinks

Als nächstes kommen wir zu den Benutzermakros für die Hyperlinks. Um einen internen Hyperlink zu einer Schlüsselstelle im Dokument zu erstellen, muss man erst ein Mal einen Anker festlegen und dann auf diesen verweisen. Mit dem Makro

```
\hypertarget{Marke}{Hier ist der verwunschene Anker.}
```

legt man den Anker fest. Mit dem

```
\hyperlink{Marke}{Click dich zum Anker durch.}
```

verweist man auf den markierten Anker.

Man muss beachten, dass mit dem *\hyperlink* nicht auf die mit *\label* markierte Stellen verwiesen werden kann. Dafür sollte man den Makro

```
\hyperref[Marke]{Text}
```

benutzen.

Um auf eine Internetseite mit einer URL zu verweisen, braucht man folgendes Makro

```
\href{URL}{Text}
```

Möchte man nocht die URL preisgeben oder anzeigen lassen braucht man nur folgendes Konstrukt anzugeben:

```
\url{URL}
```

Die Abbbildung8.1 zeigt einen Beispiel, in dem die erste Zeile mit dem Makro *\hyperref* und die zweite Zeile mit *\href* erzeugt wurden.

Sprünge in externe Dokumente aus einem aktuellen Dokument sind auch möglich. Dieses Ziel kann man mit Hilfe von xr-Package erreichen. Oder man benutzt folgendes Makro

Beispiel
Proseminar: Wissenschaftliches Arbeiten mit LaTeX
`http://homepage.mac.com/farwer/latex05/`

Abbildung 8.1: Hyperlinkverweis auf eine Internetseite und die zugehörige URL

```
\href{Ziel.pdf#Marke}{Click mich!}
```

Über den Linktext „Click mich!“ gelangt man in ein externes Dokument namens Ziel.pdf zu dem angegebenen Anker, hier „Marke“.

8.4.3 Formularumgebung

Mit dem Hyperrefpackage kann man nicht nur Hyperlinks erzeugen. Man kann unter anderem auch Formulare erzeugen, die den HTML-Formularen ähnlich sind. Ein Formular wird als ein PDF-Dokument erstellt. In diesem Dokument kann man interaktiv alle Felder ausfüllen. Mit LaTeX werden die Formulare nur erstellt, die Auswertung dieser muss man in einer anderen Sprache so wie JavaScript definieren. Es gibt vier verschiedene Arten der Formularfelder:

- Textfeld
- Kontrollkästchen
- Auswahlfelder
- Schaltflächen

Um ein Formular zu erzeugen, muss man die Formumgebung formulieren:

```
\begin{Form}
⋮
\end {Form}
```

Ein Textfeld wird mit folgendem Makro erzeugt:

```
\TextField[optionen]{Beischreibung}
```

Die Abbildung8.2 zeigt ein Bespiel für einen Textfeld, der mit dem Befehl erzeugt wurde: \TextField[width=3in,name=xname,value={Mustermann}]{dein vollständiger Na-

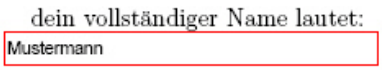


Abbildung 8.2: Textfeld mit einem Mustertext

me lautet:} Man kann die Größe des Fensters, des einzugebenden Textes und einen Mustertext bei dem Textfeld festlegen. Man kann es auch für Passworteingabe einstellen. Beispiel in der Abbildung8.3 wurde mit emph\TextField[password,name=made]{dein Geheimnis:} erstellt. Mit dem Makro

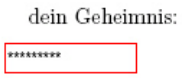


Abbildung 8.3: Textfeld für Passworteingabe

\ChoiceMenu [optionen]{Beschreibung}{Möglichkeiten}

erstellt man Auswahlkontrollkästchen. Ein Beispiel dafür zeigt die Abbildung8.4 mit dem

\ChoiceMenu[radio,default=Again,name=next,borderwidth=3,bordercolor= 0 1 0]{Kennst du Hyperref-Package?}{ja=Again,nein=Forget, vielleicht=Double} erstellt wurde. Mit

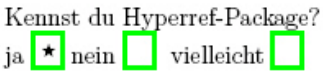


Abbildung 8.4: Auswahl-Kontrollkästchen in einem Formular

dem Makro *\CheckBox[optionen]{Beschreibung}* erstellt man Checkboxen, wie die Abbildung8.5 zeigt. Der dazugehörige Befehl lautet:

\emphWas haben wir jetzt gesehen?

\CheckBox[name=punkta,checked,bordercolor= 0 1 0]{Textfelder}

\CheckBox[bordercolor= 0 1 0]{Auswahlfelder als Kontrollkästchen}

\CheckBox[bordercolor= 0 1 0]{Kontrollkästchen}

\CheckBox[bordercolor= 0 1 0]{Schaltflächen Übergeben und Verwerfen}

\CheckBox[bordercolor= 0 1 0]{Schaltflächen mit Actionen}}

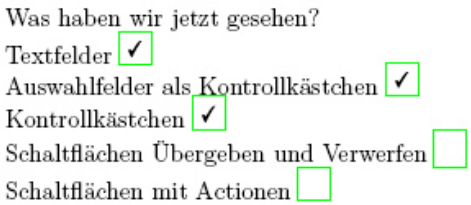


Abbildung 8.5: CheckBoxen in einem Formular

8.4.4 Fazit

Mit dem Hyperrefpackage kann man, wie wir es gesehen haben, die Forderungen, die ein internetfähiges Dokument erfüllen muss, erreichen. Man kann mit dem Hyperrefpackage interne und auch externe Hyperlinks und Formulare zum interaktiven Ausfüllen erzeugen.

Das Hyperrefpackage kann viel mehr leisten als es hier gezeigt wurde. Unter anderem kann man eigenen Lesezeichen-Code erzeugen oder automatisch erzeugen lassen. Mehr Informationen zur praktischen Anwendung finden Sie unter `http://www.ibnm.uni-hannover.de/Mitarbeiter/beuerman/LaTeX2PDF.pdf`.

Kapitel 9

Erweiterte Dokumentklassen und Packages

Felix Fietkau, Volker Lukas

9.1 Das Key-Value Interface for Optionen

Heutzutage benutzen zahlreiche LaTeX-Pakete das Paket "keyval", welches eine komfortable und flexible Methode zum Angeben von Argumenten für Kommandos bietet. Dazu gehören auch die in diesem Kapitel vorgestellten Pakete. Deshalb soll hier kurz auf die Syntax eingegangen werden, die vom Keyval Paket benutzt wird.

Das Grundproblem mit der bei LaTeX eingebauten Art des Angebens von Paramtern, besteht darin, dass die Anzahl der Parameter eine Konstante sein muss. Keyval erlaubt es, dieses Defizit zu umgehen. Dabei werden mehrere Argumente für das Kommando in einem einzigen LaTeX-Argument untergebracht. Folgende Syntax wird dafür verwendet: Die einzelnen Argumente werden durch Kommas getrennt. Ein Argument kann ein Wertepaar sein. Dabei werden die beiden Werte durch ein Gleichheitszeichen getrennt. Der erste Wert eines Argumentes wird als "Key" bezeichnet. Ist ein zweiter Wert vorhanden, so wird dieser "Value" genannt. Wird ein Teil des Wertepaares von geschweiften Klammern umschlossen, dann wird das äußerste Klammerpaar von Keyval entfernt, und der Inhalt des Klammerpaares an das Kommando weitergegeben. So ist es möglich, Argumente, die Kommas oder Gleichheitszeichen enthalten, an ein Kommando zu übergeben. Davon abgesehen werden Argumente exakt wie sie erscheinen von Keyval weitergegeben. Dies bedeutet, dass etwas, das wie der Aufruf eines Kommandos aussieht, wenn es in

der Keyval Argumentenliste erscheint, nicht ausgewertet wird. Ein Auswertung würde erst dann stattfinden, wenn das Kommando, welches das Key-Value Interface verwendet, seinerseits das erhaltene Argument als Kommando ausführt.

9.2 KOMA-Script

Ein umfangreiches Paket für LaTeX ist KOMA-Script, welches unter anderem Dokumentklassen für Artikel, Reports, Bücher und Briefe enthält. Trotz des großen Umfangs wird hier jedoch nur die Briefklasse von KOMA-Script vorgestellt.

9.2.1 Die Motivation von KOMA-Script

Die in LaTeX eingebauten Dokumentenklassen sind hauptsächlich für englischsprachige Dokumente gemacht worden. Um z.B. deutsche Konventionen einzuhalten, ist ein gewisser Aufwand zu betreiben. KOMA-Script ist ein Paket, welches Vorlagen für speziell deutschsprachige Dokumente beinhaltet. Der Inhalt beschränkt sich jedoch nicht auf eine simple Adaptierung der originalen Dokumentenklassen, vielmehr besticht KOMA-Script durch eine Fülle von Formatierungsoptionen, durchdachter Unterstützung von typographisch ansprechenden Dokumenten, und einigen nützlichen Utilities.

9.2.2 Ein Abriss der KOMA-Script Geschichte

Die Wurzeln von KOMA-Script reichen zurück bis in die frühen 1990er Jahre. Damals musste Frank Neukamm ein Vorlesungsskript setzen. Die damals erhältlichen Styles für LaTeX stellten ihn jedoch nicht zufrieden. Das bewegte ihn dazu, seine eigenen zu schreiben. Diese veröffentlichte er unter dem Namen “Script”.

1992 fand Markus Kohm “Script”, welches er erweiterte. Im Jahre 1993 veröffentlichte Frank Neukamm eine zweite Version von “Script”, namens “Script-2”. Diese wurde von Markus Kohm auf das damals neue LaTeX 2_ε portiert. Schließlich stellte er dieses Paket der

ffentlichkeit zur Verfügung. Seit dieser Version trägt das Paket den Namen KOMA-Script.

Seit der ursprünglichen Veröffentlichung fand das Paket breiten Anklang, so dass Markus Kohm als Entwickler nicht alleine geblieben ist.

9.2.3 Die Briefklasse von KOMA-Script

Eine der am stärksten von den Standard-LaTeX-Klassen abweichende KOMA-Script Klasse ist "scrlttr2", die Briefklasse. Folgende Schritte benutzen die Klasse um das Gerüst für ein Briefdokument zu erstellen:

```
% KOMA Briefklasse laden, Papier ist A4
\documentclass[paper=a4]{scrlttr2}
\usepackage[latin1]{inputenc}
\usepackage[ngerman]{babel}
% Laden von Einstellungen für DIN Briefgestaltung.
\LoadLetterOption{DIN}

\begin{document}
\begin{letter} %
Öffnen der Briefumgebung
 {\\
 To address % Zweites Argument ist die Empfängeraddresse
 }
 \opening{Gruß} % Dieses Kommando muss folgen

 % Hier folgt der Textkörper des Briefes.
 Kaum Text hier.

 \closing{Gruß} % Abschließende Grußfloskel

\end{letter}
\end{document}
```

Zunächst ist die Klasse zu laden. Das Kommando LoadLetterOption gehört zu KOMA-Script. Hierdurch werden einige Einstellungen geladen, die das Formatieren des Briefes

entsprechend des Arguments ermöglichen. In diesem Fall also nach DIN. Ein KOMA-Script Brief wird innerhalb des Environments "document" durch ein weiteres Environment namens "letter" geschrieben. Zu beachten ist, dass beim ffnen des Letter-Environment als zweites Argument die Adresse des Empfängers folgt. Die einzelnen Felder der Adresse, also Namen, Straße und Hausnummer, sowie Postleitzahl und Ort, werden hierbei durch doppelte Backslashes getrennt. Damit der Briefkopf gesetzt wird, muss innerhalb dieses Environments das Kommando "opening" benutzt werden. Argument ist hierbei die einleitende Grußformel. Nach diesen Vorarbeiten kann dann der restliche Text des Briefes folgen. Ein Brief sollte mit Aufrufen des "closing" Kommandos beendet werden, wobei dieses Kommando noch eine abschließende Floskel enthalten kann.

Die Briefklasse kennt so genannte Variablen, welche Einstellungen fassen, die das Aussehen, aber auch den Inhalt des Dokumentes beeinflussen. Diese Variablen können wie folgt belegt werden:

```
\setkomavar{Variablenname}{Wert}
```

Einige wichtige Variablen sind "fromname", die den Namen des Absenders fasst, "fromaddress" für die Adresse (ohne den Namen) des Absenders, sowie der Betreff in "subject", und gegebenfalls die Email-Address des Absenders in "fromemail". Für eine Auflistung aller Variablen sei auf die KOMA-Script Dokumentation verwiesen [KM04].

Um das Aussehen und den Inhalt des Briefes zu gestalten, können durch das Kommando "KOMAoptions" optionale Einstellungen getätigt werden. Eine Option beginnt dabei mit einem Namen, auf den nach einem Gleichheitszeichen noch ein Wert folgen kann. Ein Beispiel wäre

```
\KOMAoptions{fromemail}
```

Dies würde bedeuten, dass das Briefdokument, spezieller der Briefkopf, die Absender-Email enthält. Wie man vielleicht erahnen kann, stehen einige der hier angegeben Optionen mit den oben angesprochenen Variablen in Zusammenhang. Zum Beispiel würde hier durch setzen der *Option* "fromemail" der Inhalt der *Variablen* "fromemail" benutzt werden, um die tatsächliche Emailadresse zu erhalten. Weitere Optionen die angegeben werden können sind "numericaldate", die festlegen würde, dass das Datum in einer rein

nummerischen Form dargestellt werden soll, “fromphone” und “fromfax” die steuern ob die Telefon-, respektive Faxnummer gesetzt wird, und “backaddress”, bei der nach einem Gleichheitszeichen angegeben werden kann, wie die Rücksendeadresse lautet. Häufig ist die Option “fromalign” von Interesse, die steuert, wo die Absenderadresse gedruckt wird. Werte sind “left”, “center” und “right”, welche die Adresse links, in der Mitte beziehungsweise rechts setzen. Fromalign steuert auch, ob KOMA-Script die zahlreichen Optionen zur Gestaltung des Briefkopfes aktiviert. Wird “off” angegeben, dann stehen nicht alle Optionen zur Verfügung.

Gerade in deutschsprachigen Briefen wird auch die Option “parskip” benutzt. Sie steuert den vertikalen Abstand zwischen den Absätzen. Der bei KOMA-Script voreingestellte Wert ist, dass kein zusätzlicher vertikaler Abstand vorhanden ist. Dafür beginnen Absätze mit einer Einrückung. Die Entwickler von KOMA-Script raten dazu, diese Einstellung auch beizubehalten. Trotzdem kann mit Absatzabständen gebarbeitet werden, wenn dies gewünscht ist. Dazu wird als Wert für “parskip” “full” angegeben, um eine ganze Zeile Abstand zu erhalten, oder “half”, für eine halbe Zeile. Diese Regeln kennen auch noch Variationen, so dass gesteuert werden kann, zu welchem Teil die letzte Zeile eines Absatzes ausgenutzt werden darf.

Um eine umfassende Referenz von Optionen kennenzulernen, sollte hier ebenfalls die Original-Dokumentation zu Rate gezogen werden.

Eine weitere Besonderheit von KOMA-Script ist, dass es eine eigene Konvention zum Laden von Einstellungen in Dokumenten-externen Dateien gibt. Dies sind die so genannten “Letter Class Option”, kurz “lco” Dateien. In diese Dateien kann der Benutzer immer wiederkehrende Angaben wie seine Adresse und Email setzen (dazu können die entsprechenden KOMA-Script Kommandos benutzt werden). Mit dem oben bereits verwendeten LoadLetterOption Kommando kann eine solche Datei eingebunden werden. Als Argument wird dabei der Namen der Datei verwendet, allerdings ohne die für solche Dateien übliche Endung “.lco”. Natürlich wäre es auch möglich, über Standard-LaTeX Anweisungen wie “input” externe Dateien einzulesen. Ein Vorteil von lco-Dateien ist jedoch, das KOMA-Skript speziell für deren Verwendung vorbereitet ist, und es so z.B. möglich ist, bereits mit Laden der “scrlttr2” Klasse eine Lco-Datei mitzuberücksichtigen.

9.3 Das Paket “Listings”

9.3.1 Listings, ein Pretty-Printer für Sourcecode

Das Paket Listings wurde 1996 von Carsten Heinz erstmals veröffentlicht. Listings ist ein Pretty-Printer für Sourcecode aller Art. Zu den Features zählen die Unterstützung für zahlreiche Sprachen, unter ihnen Ada, C++, Delphi, HTML, Lisp, SQL und XML. Diese Liste ist bei weitem nicht vollzählig, wie ein Blick in die Originaldokumentation verrät [Hei02]. Die Anzeige des Codes kann durch viele Einstellungen geregelt werden. Unter anderem ist es möglich, verschiedene grammatikalische Elemente der darzustellenden Sprache, unterschiedlich formatiert auszugeben, und es ist möglich die Ausgabe mit Zeilennummern und Rahmen zu versehen.

9.3.2 Grundsätze der Benutzung von Listings

Das durch das usepackage Kommando einzubindende Paket lautet “listings”. Dann können “lstlisting”-Environments angelegt werden, in denen der zu formatierende Code direkt einzugeben ist. Optionen können durch Aufrufen von “lstset” angegeben werden. Folgende Optionen gehören zu den am häufigsten gebrauchten: “extendedchars=true” wird gebraucht, um Zeichen, deren nummerischer Zeichencode höher als 127 ist, korrekt einzulesen. “basicstyle=*Wert*” legt den Stil für “normale” Elemente des Codes fest. Die Werte für diesen Parameter können Schriftgrößen wie \scriptsize sein, oder Fontfamilien wie beispielsweise \ttfamily. Neben dem kann unter anderem auch die Schriftfarbe ausgewählt werden. Weitere Sprachelemente, die jeweils unterschiedlich formatiert werden können, sind Kommentare (“commentstyle=*Wert*”), Schlüsselwörter (“keywordstyle=*Wert*”) und Strings (“stringstyle=*Wert*”). Diese Parameter erwarten wiederum Werte aus der gleichen Menge wie für “basicstyle”.

Als vermutlich wichtigstes Argument sei hier noch die Einstellung der Sprache erwähnt, “language=*Wert*” wählt hier *Wert* als Sprache aus.

Ein Beispiel wird noch verdeutlichen, wie das Setzen von Optionen konkret aussehen kann.

9.3.3 Escaping von nicht *verbatim* darzustellenden Eingaben

Listings bietet eine hilfreiche und einfach zu benutzende Möglichkeit, einen Teil der Eingabe von der Darstellungsweise des übrigen Codes auszunehmen, und mit diesem Teil besonders zu verfahren. Dazu reicht es aus, bei "lstset" die Option "mathescape" anzugeben. Auf diese Weise werden Teile der Eingabe zwischen einem Paar von Dollarzeichen, wie sie in LaTeX ein Math-Environment abgrenzen, nicht wie der Rest des Codes behandelt. Dieser Teil der Eingabe wird vielmehr LaTeX üblich verarbeitet, das heißt LaTeX-Kommandos werden wahrgenommen und können z.B. zur Änderung der Formatierung verwendet werden. Das Dollarzeichen eignet sich natürlich in besonderer Weise als Escapezeichen, weil es weniger häufig als andere Zeichen in Texten vorkommt.

Listings kann darüber hinaus auch andere Zeichen als Escapezeichen erkennen. Wie diese zu definieren sind, kann der Originaldokumentation entnommen werden.

9.3.4 Einlesen externer Dateien

Die zweite Möglichkeit Code einzugeben, der durch Listings dargestellt werden soll, besteht darin, eine externe Datei einzulesen. Dazu dient das Kommando "lstinputlisting", welches als Argument den Namen der einzulesenden Datei erwartet.

9.3.5 Zwei kurze Beispiele zu Listings

Das erste Beispiel zeigt, wie ein Codefragment verbatim dargestellt werden kann.

```
\documentclass{article}
\usepackage{listings}

\begin{document}
\begin{lstlisting}
#include <iostream>

int main(char* argc, char* argv[])
{
std::cout << "\nHello World\n";
```

```
}

\end{lstlisting}
\end{document}
```

Das zweite Beispiel verwendet das gleich Codefragment wie im ersten Fall, jedoch wurde die Darstellung verändert.

```
\documentclass{article}
\usepackage{listings}

\begin{document}
\lstset{language=C++,numbers=left}
\begin{lstlisting}
#include <iostream>

int main(char* argc, char* argv[])
{
std::cout << "\nHello World\n";
}

\end{lstlisting}
\end{document}
```

9.4 Das “Fancyvrb” Paket

9.4.1 Aufgaben für Fancyvrb

Im letzten Abschnitt haben wir Listings kennengelernt, welches sich mit dem Formatieren von Code befasst. Wie wir gesehen haben, ist die Darstellung von Code, ohne dabei auf besondere Formatierungsmöglichkeiten zurückzugreifen, sehr ähnlich einer Darstellungs des Textes *verbatim*. Zu diesem Zweck gibt es aber ein Package welches sich speziell damit befasst. Dies ist Fancyvrb. Ursprünglich von Timothy van Zandt entwickelt, wird es nun von Dennis Girou und Sebastian Rahtz weiterentwickelt.

9.4.2 Einbinden von Fancyvrb und Grundsätzliches

Das Paket zum einbinden heißt "fancyvrb". Besondere Optionen zum Laden sind nicht nötig.

Der darzustellende Text kann in einem LaTeX-Environment direkt im Dokument eingegeben werden. Es ist auch möglich, in einem speziellen Environment den Text einzugeben, und an einer anderen Stelle im Dokument den Inhalt auszugeben, oder den Text aus einer Datei einlesen zu lassen.

Verwendet man die erste Möglichkeit, folgt ein Dokument also diesem Muster:

```
\usepackage{fancyvrb}
...
\begin{Verbatim}
Der Text steht hier.
\end{Verbatim}
```

Wie man sehen kann, heißt das Environment in dem der Text eingegeben wird "Verbatim".

9.4.3 Abschneiden des Zeilenanfangs

Manchmal kann es nützlich sein, eine feste Anzahl von Buchstaben am Anfang jeder Zeile abzuschneiden, z.B. wenn der Text in jeder Zeile mit Leerzeichen beginnt, oder wenn zum Beispiel bereits eine Zeilennummerierung besteht, die nicht übernommen werden soll. Dieses Abschneiden wird auf Englisch auch *to gobble* genannt. Daher trägt die Option,4 die beim ffnen des Verbatim-Environment angegeben wird, den Namen "gobble".

```
\begin{Verbatim}[gobble=4]
Der Text steht hier.
\end{Verbatim}
```

Hieraus ist auch ersichtlich, wo Fancyvrb die Angabe von Optionen erwartet: Sie folgen als optionaler Parameter direkt auf das Kommando zum ffnen des Environment.

9.4.4 Weitere Möglichkeiten der Darstellung

Zu oft nützlichen Methoden die Darstellung anzupassen, gesellen sich Rahmen um den Text, die Nummerierung von Zeilen, und gelegentlich auch die farbliche Gestaltung. Ein Beispiel soll veranschaulichen, wie dies mit Fancyvrb möglich ist.

```
\begin{Verbatim}[frame=single,numbers=left,rulecolor=\color{green}]
Der Text steht hier.
\end{Verbatim}
```

Die Angabe "frame=single" bewirkt hier die Ausgabe eines Rahmens, in diesem Fall als einfache Linie. "numbers=left" legt fest, dass am linken Rand Zeilennummern erscheinen. Schließlich färbt "rulecolor=\green" noch den Rahmen grün. Stellt man auf diese Art Farben ein, ist im LaTeX-Dokument das "color" Paket zu laden.

9.4.5 Zwei Möglichkeiten indirekt Text einzugeben

Um Text in einem Environment zu speichern, kann dazu das Environment "SaveVerbatim" benutzt werden. Dieses erwartet als Argument den Namen unter dem der Text angesprochen werden soll. Dort wo das Environment im LaTeX-Code auftaucht geschieht keine Ausgabe. Diese wird erst generiert wenn über das Komando "UseVerbatim" das Environment benutzt wird. UseVerbatim erhält als Argument den zuvor vergebenen Namen.

```
\begin{SaveVerbatim}{EinName}
Hier kann der Text stehen.
\end{SaveVerbatim}
...
\AnderesEnvironment
 {
 ...
 \UseVerbatim{EinName}
 ...
 }
```

Optionen zur Darstellung können auf das UseVerbatim Kommando folgen. Manche Optionen werden jedoch bereits beim ffnen des SaveVerbatim-Environment erwartet. Zur Veranschaung hier das letzte Beispiel etwas verändert:

```
\begin{SaveVerbatim}[gobble=5]{EinName}
Hier kann der Text stehen.
\end{SaveVerbatim}
...
\AnderesEnvironment
 {
 ...
 \UseVerbatim[frame=single,label=Ein Text]{EinName}
 ...
 }
```

Es besteht auch noch die Möglichkeit, Text aus einer Datei zu lesen. Dazu dient das Kommando “VerbatimInput”. Dieses Kommando kann als Optionales Argument eine Reihe von Optionen erhalten, und muss in jedem Fall ein Argument mit dem Dateinamen erhalten.

```
\VerbatimInput[Hier optionale Angaben]
 {Dateiname}
```

9.5 Abkürzungen mit “acronym”

Ein kleines aber dennoch nützliches Paket ist Tobias Oetikers “acronym”. Acronym hilft dabei, dass in einem Dokument nicht vergessen wird, mindestens einmal jede Abkürzung auszuschreiben. Einbinden des Paketes geschieht wie üblich durch \usepackage{acronym}. Als Option kann hier noch “footnote” angegeben werden. Geschieht dies, werden die ausgeschriebenen Formen von Akronymen in Fußnoten gesetzt. Andernfalls geschieht dies direkt im Text.

Um ein Akronym und seine Erklärung zu definieren, kann ein Environment mit dem Namen “acronym” geöffnet werden. In diesem Environment können mit dem Kommando “acro” Akronyme angegeben werden. Dieses Kommando erwartet als erstes Argument

die Kurzform und als zweites die ausgeschriebene Form. Eine Erklärung kann einem so definierten Akronym folgen, allerdings nicht als Argument des acro Kommandos, sondern hinter den Aufruf des Kommandos geschrieben. Das acronym Environment wird im Dokument mitausgegeben. Es ist so nicht nur zum Definieren geeignet, sondern dient auch gleichzeitig als eine Art Glossar für Abkürzungen. Will man dieses nicht haben, dann kann man auch mit dem Kommando "acrodef" Akronyme definieren. Dieses Kommando muss nicht in einer besonderen Umgebung stehen, und an der Stelle an der es verwendet wird, wird auch keine Ausgabe erzeugt. Wie auch das acro Kommando, erwartet acrodef als erstes Argument die Kurz- und als zweites die Langform.

Um Akronyme im Text zu verwenden, dienen drei Kommandos.

```
\ac{Kurzform}
```

Dieses Kommando überprüft, ob das Akronym mit der angegebenen Kurzform bereits verwendet wurde. Wenn ja, dann wird nur die Kurzform gesetzt. Wenn nicht, dann wird die Langform mitausgegeben.

```
\acf{Kurzform}
```

Das Kommando acf erzwingt die Ausgabe der Langform. Dieses Kommando wird komplementiert durch acs, welches nur die Kurzform ausgibt.

```
\acs{Kurzform}
```

Das folgende Beispiel erklärt die Akronyme in Fußnoten, und enthält am Ende ein kleines Glossar:

```
\documentclass[a4paper]{article}

\usepackage[footnote]{acronym}
\begin{document} \paragraph*{}{

\acrodef{EA}{Ein Akronym}
Hier wird ein Akronym das erste mal verwendet \ac{EA}.
Und hier ein zweites Mal \ac{EA}. Und jetzt ein anderes Akronym: \ac{EWA}
```

```
\paragraph*{\Large Glossar:}{

\begin{acronym}
\acro{EWA}{Ein Weiteres Akronym} - Dies ist eine Erklärung.

\end{acronym} }

\end{document}
```

In dem Beispiel kann man sehen, dass die Definition eines Akronyms im LaTeX-Code nicht vor der Verwendung stehen muss. Allerdings sind in einem solchen Fall zwei LaTeX Durchläufe nötig, damit Acronym die Definition verwenden kann.

Hier noch ein Beispiel, welches auf Fußnoten verzichtet:

```
\documentclass[a4paper]{article}

\usepackage{acronym}
\begin{document} {

\acrodef{EA}{Ein Akronym}
Hier wird ein Akronym das erste mal verwendet \ac{EA}.
Und hier ein zweites Mal \ac{EA}

\end{document}
```

9.6 Erstellen von eigenen Dokumentklassen und Packages

9.6.1 Klasse oder Package?

Wenn man eine LaTeX-Erweiterung erstellen möchte, muss man sich zuerst entscheiden, ob es eine Klasse oder ein Package werden soll.

Ein Package (Dateiendung `.sty`) wird von einem Dokument, einem anderen Package oder einer Klasse aus geladen. Die Funktionalität die es bereitstellt ist üblicherweise nicht an ein bestimmtes Format gebunden.

Eine Klasse (Dateiendung `.cls`) gibt für ein Dokument das Layout vor. Dabei kann sie auch eine andere Klasse als Basis nehmen und bestimmte Eigenschaften ändern.

9.6.2 Integrieren einer LaTeX-Erweiterung

Klassen und Packages müssen, damit sie vom LaTeX-System gefunden werden, in die `texmf`-Verzeichnisstruktur eingebunden werden, welche alle plattformunabhängigen, zum TeX-System gehörenden, Dateien enthält. `texmf` steht dabei für "TeX and Metafont" - die typische Basis einer normalen TeX-Installation.

Normalerweise gibt es auf einem System mehrere `texmf`-Verzeichnisse, unterteilt in:

- Hauptverzeichnis für `texmf`
- Andere lokale `texmf`-Verzeichnisse
- Benutzerspezifische `texmf`-Verzeichnisse

Wo sich diese Verzeichnisse befinden ist abhängig von dem Betriebssystem und der verwendeten TeX-Distribution.

Bei den zwei am häufigsten verwendeten Distributionen kann man sich den Suchpfad für `texmf`-Verzeichnisse so ausgeben lassen:

- teTeX (UNIX/-kompatibel) kennt das Kommando `kpsepath tex`,
- MiKTeX (Windows) hat dafür ein Konfigurationsprogramm (im Startmenü)

Damit die Installation von vielen Packages und Klassen auf einem System funktioniert, ohne dass diese sich gegenseitig stören, wurde die Struktur der `texmf`-Verzeichnisse vereinheitlicht. Dazu gibt es ein Standard-Dokument: "The TeX Directory Structure" [v1.04], herausgegeben von der TDS Working Group, welches genau regelt, wie die Makros/Pakete/Klassen/Fonts in den `texmf`-Verzeichnissen unterteilt sind.

Dabei wurde vor allem darauf geachtet, dass verschiedene TEX-Implementationen auf verschiedenen Plattformen untereinander kompatibel sind, so dass sich die Autoren von Packages nicht auf implementationsabhängige Abweichungen einstellen müssen.

Beispiele für TDS-Hauptverzeichnisse:

- `tex/` - enthält alle Dateien, die direkt vom TEX-System verarbeitet werden
- `bibtex/` - enthält verschiedene BibTEX-Dateien
- `fonts/` - enthält Fonts für das TEX-System in verschiedenen Formaten
- `scripts/` - enthält plattformunabhängige Scripts, die vom TEX-System verwendet werden können

Das `tex/` Verzeichnis ist für verschiedene TEX-Systeme (wie z.B. LATEX) und für einzelne Packages unterteilt. Beispiel: `tex/latex/`*`beamer/`* für das *beamer* Package.

9.6.3 Aufbau von Klassen und Packages

Klassen und Packages bestehen normalerweise aus vier Teilen:

- **Identifikation**:
  Hier steht, welche LATEX-Version erforderlich ist (normalerweise LATEX 2_ε), sowie Name und Erstellungsdatum von dem Package oder der Klasse
- **Vorläufige Deklarationen**:
  Hier stehen alle Deklarationen, die für das Verarbeiten von Optionen notwendig sind
- **Optionen**:
  Hier steht der LATEX-Code für das Verarbeiten von Optionen
- **Hauptteil**:
  Im Hauptteil befindet sich dann der wesentliche Code für das Package bzw. die Klasse. Hier können auch weitere Packages eingebunden werden.

9.6.4 Beispiel

Hier ein Beispiel-Package, welches das UNIX-Kommando 'fortune' in LaTeX integriert (`example.sty`):

```
% Identifikation
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{example}[2005/06/23 My First Package]

% Vorläufige Deklarationen
\newcommand{\fortunetype}{}
\newcommand{\fortunelen}{}

% Optionen
\DeclareOption{1line}{\renewcommand{\fortunelen}{"-n1"}}
\DeclareOption{2line}{\renewcommand{\fortunelen}{"-n2"}}
\DeclareOption{off}{\renewcommand{\fortunetype}{"off"}}
\DeclareOption{wisdom}{\renewcommand{\fortunetype}{"wisdom"}}
\ProcessOptions\relax

% Hauptteil
\newcommand{\fortune}{
        \immediate\write18{
                fortune \fortunelen \fortunetype > fortune.tex
        }
        \input{fortune}
}
```

Dieses Package kann man dann wie in diesem Beispiel verwenden (`example.tex`):

```
\documentclass{article}

\usepackage[1line,wisdom]{example}

\begin{document}
```

```
        \subsection*{Fortune Cookie of the Day}
        \Large \fortune
\end{document}
```

9.6.5 Verfügbare Befehle

Die Liste der Befehle, die hier aufgeführt sind, enthält nur die wichtigsten Befehle, um einfache Packages oder Klassen zu erstellen. Weitere Befehle und Erläuterungen befinden sich im Dokument "*LaTeX 2ε for class and package writers*" [Pro99], welches auch die Konzepte zur Erstellung von Packages näher erläutert.

Alle Befehle die speziell für das Schreiben von Packages oder Klassen entworfen wurden beginnen mit einem Großbuchstaben nach dem Backslash. Durch diese Konvention kann man sie leicht von regulären LaTeX-Befehlen unterscheiden. Einige der Befehle sind von der Funktionsweise her wie normale LaTeX-Befehle, haben aber speziell für Packages und Klassen neue Namen bekommen.

- `\NeedsTeXFormat` - Legt die benötigte LaTeX-Version fest
  - Beispiel: `\NeedsTeXFormat{LaTeX2e}`
- `\ProvidesPackage` - Diese Datei enthält ein Package.
  Als erstes Argument wird der Package-Name übergeben, als optionales Argument kann noch das Datum, sowie eine Beschreibung mit angegeben werden. Das Datum muss in dem Format JJJJ/MM/TT sein und unbedingt mit angegeben werden, falls das optionale Argument verwendet wird.
  - Beispiel: `\ProvidesPackage{example}[2005/06/23 My First Package]`
- `\ProvidesClass` - Diese Datei enthält eine Klasse (siehe `ProvidesPackage`).
  - Beispiel: `\ProvidesClass{example}[2005/06/23 My First Class]`
- `\RequirePackage` - Bindet ein externes Package ein (wie `\usepackage`). Kann mehrmals für verschiedene Packages aufgerufen werden.
  - Beispiel: `\RequirePackage{babel}`

- `\LoadClass` - Bindet eine Dokumentklasse als Vorlage ein (wie `\documentclass`). Wird nur in Dokumentklassen verwendet - dabei kann nur eine einzige Klasse auf diese Weise als Vorlage verwendet werden.
  - Beispiel: `\LoadClass{article}`
- `\DeclareOption` - Deklariert eine Option für das Package oder die Klasse.
  Das erste Argument enthält den Namen der Option, das zweite Argument den Code, der für diese Option ausgeführt werden soll.
  - Beispiel: `\DeclareOption{1line}{\renewcommand{\lines}{1}}`
- `\PassOptionsToPackage` - Gibt die angegebenen Optionen an ein Package weiter. Das erste Argument enthält die Liste der zu übertragenden Optionen, das zweite Argument enthält den Namen des Package, das die Optionen erhalten soll.
  - Beispiel: `\PassOptionsToPackage{german}{babel}`
- `\PassOptionsToClass` - Gibt die angegebenen Optionen an eine Dokumentklasse weiter (siehe `\PassOptionsToPackage`)
  - Beispiel: `\PassOptionsToClass{a4paper}{article}`
- `\ProcessOptions` - Verarbeitet die Optionen. Es wird dabei empfohlen, den Befehl `\relax` mit anzugeben.
  - Beispiel: `\ProcessOptions\relax`

Kapitel 10

Wie geht man ein wissenschaftliches Projekt an?

Annika Rosner, Felix Dobslaw

10.1 Wissenschaften

10.1.1 Definiton

Die klassischen Definitionsregeln gehen auf Aristoteles zurück (Vgl. Analytica Posteriora, Organon) (zit. nach Kondakow 1983, S. 81) [Wik05]:

1. Ein Begriff wird durch seine nächste Gattung und den Artunterschied definiert (Praecisio definitionis). (veraltet)
2. Der Artunterschied muss ein Merkmal oder eine Gruppe von Merkmalen sein, die nur dem vorliegenden Begriff zukommen und bei anderen Begriffen fehlen, die zur selben Gattung gehören. (veraltet)
3. Eine Definition muss angemessen sein, d.h. weder zu weit noch zu eng gefasst sein.
4. Eine Definition darf keinen Zirkelschluss enthalten.
5. Eine Definition darf keine logischen Widersprüche enthalten.
6. Eine Definition darf nicht nur negativ bestimmt sein

7. Eine Definition darf keine Mehrdeutigkeiten enthalten.

Selbst nach diesen Regeln ist es schwer für *Wissenschaft* eine Definition aufzubauen. Trotzdem wollte ich versuchen eine zu finden, hier ist sie:

Wissenschaft ist das Wissen einer Zeit, welches durch Forschen, Lehre und überlieferter Literatur gebildet, geordnet und begründet wurde. [Div97]

Ich gebe zu, dass ich gegen den Punkt 3 der Definitionsregeln verstosse, denn es gibt zu viele Merkmale, Eigenschaften und Ziele der Wissenschaft, dass man es lieber in einem Lexikon nachschlagen sollte.

10.1.2 Teilgebeite der Wissenschaft

- Humanwissenschaften
- Geisteswissenschaften
- Humanwissenschaften
- Ingenieurwissenschaften
- Naturwissenschaften
- Philosophie und Religionswissenschaften
- Sozialwissenschaften
- Strukturwissenschaften
- Wirtschaftswissenschaften
- Wissenschaftstheorie

Meist wird die Geisteswissenschaften auch Kulturwissenschaften genannt. Auch wird des öfteren die Wissenschaft nicht in so viele Bereiche unterteilt, sondern viel auch nur in Geisteswissenschaft und Naturwissenschaft. Doch dabei sieht man schnell dass man viele Bereiche nicht mit diesen beiden Wissenschaften identifizieren kann. Informatik und Mathematik kann man eigentlich nicht so wirklich in eine der beiden Wissenschaften packen, daher gehören diese Beiden auch zu den Strukturwissenschaften.

10.1.3 Drei Wissenschaften

Womit wir uns jetzt am meisten auseinandergesetzt haben waren die drei Wissenschaften die wir hierfür am wichtigsten angesehen haben: Natur-, Geistes- und Strukturwissenschaft.

Die Naturwissenschaft [Wik05] beschäftig sich mit der belebten und unbelebten Natur, hierzu gehören unter anderem Physik, Biologie und Chemie. Man versucht alle Erscheinungen und deren Ursache zu erklären, geht aber davon aus dass es nichts übernatürliches oder willkürliches in der Natur gibt und alle Versuche wiederholbar sind. In dieser Wissenschaft werden Theorien meist in Mathematischer Form verfasst.

Die Mathematik gehört wie schon geschrieben zur Strukturwissenschaft [Wik05]. Der Begriff Strukturwissenschaft wurde 1971 von Carl Friedrich von Weizsäcker geprägt. Diese Wissenschaft wird auch als Bindeglied zwischen der Natur- und Geisteswissenschaft angesehen. Die Strukturwissenschaft befasst sich nicht mit der Erforschung von Gegebenheiten, sondern mit der Vorangehensweise neuer Erkenntnisse, d.h. man hat Hypothesen und versucht dadraus eine Theorie zu entwickeln.

Die Geisteswissenschaft [Wik05] befasst sich mit der kulturellen und geistigen Schöpfung von Wissen. Das heißt nicht, dass der individuelle Geist gemeint ist, sondern einem sogenannten objektiven Geist.

10.2 Projekte

Die folgende Struktur zur Herangehensweise an Projekte (wobei Projekt noch definitionsbedürftig sei) ist hauptsächlich auf Teil 1 des Werkes „Geniale Projekte “ von Christina Maria Kunz-Koch zurück zu führen.

Zur Einführung ins Thema wird erst einmal die Projektdidaktik eingebracht, welche sich mittlerweile als Fachübegreifende Disziplin in den Wissenschaften etabliert hat. Anhand der Projektdidaktik, welcher das Buch zugrunde liegt, wird auf die 3 zu unterscheidenden Ebenen der Projektorganisation eingegangen.

Unser Hauptaugenmerkt richtet sich dabei auf die erste Ebene, die *Projektebene*. Auf dieser Ebene beschäftigen wir uns mit dem, *was* es in einem Projekt zu erledigen gilt. Das Kapitel gibt Hilfestellung zur kontrollierten Projektdurchführung, anhand einer Projekttreppe.

Im letzten Abschnitt „Geniale Projekte, Schritt für Schritt entwickeln“ [KK01] wird noch auf Ebene 2 und 3 der Zielsetzungen von Projekten eingegangen.

Im weiteren Verlauf dieses Kapitels aufgeführte Grafiken sind dem Buch entnommen.

10.2.1 Projektdidaktik

Da wir uns im Rahmen dieses Kapitels mit Projektdidaktik beschäftigen, möchten wir zur Einstimmung ins Gebiet eine Erläuterung der Begriffe Didaktik und didaktischer Modelle voraus schicken, so dass daraufhin der Begriff der Projektdidaktik, mit ausreichendem Verständnis seiner Teilbegriffe, erfasst werden kann.

Didaktik

Da das Feld der Didaktik ein von vielen Meinungen und Definitionen durchwachsenes ist, möchten wir uns bei der Definition nicht zu weit aus dem Fenster lehnen und uns an Grundsätze halten, die die Didaktik ausmachen.

Als Unterdisziplin der Pädagogik beschäftigt sich die *Didaktik* mit der „Lehre vom Lernen“, also der Vermittlung von Lerntechniken. Dies vollzieht sie unabhängig von Lerninhalten, auf die sie angewandt wird.

Es gibt eine Vielzahl von *didaktischen Modellen*, die bestimmte Zielsetzung in den Vordergrund der Didaktik stellen. Als Beispiel zu nennen sei die Bildungstheoretische Didaktik, die sich die Bildung des Menschen im ganzen als Ziel setzt.

Aha, und was ist nun Projektdidaktik?

Die *Projektdidaktik* ist nunmehr eine Unterklasse der Didaktik, die sich speziell mit der Vermittlung von Lerntechniken auf dem Gebiet von Projekten beschäftigt. Was Projekte zu etwas so Besonderem macht, dass sie eine eigene Teildisziplin der Didaktik spendiert bekommen, wird in der nächsten Sektion behandelt.

Im weiteren Verlauf beschäftigen wir uns mit dem in der Einleitung angesprochenen *projektdidaktischen Modell*, welches auf die Vermittlung von Lerntechniken auf drei Ebenen (Projekt-, Strategie- und Zielebene) beruht.

10.2.2 Was ist ein Projekt?

Das Wort *Projekt*, welches aus dem lateinischen ins Deutsche aus dem Wort „proiectum = das nach vorn Geworfene“ [KK01] hergeleitet wird, begreifen wir im folgenden als ein „vielschichtig, gemeinschaftliches Unternehmen, ..., das von einer Leitfrage zu einem Produkt (bzw. an ein Ziel) führt“ [KK01].

Für einen Bergsteiger sei somit die Leitfrage „Wie erklimme ich Berg X“. Das erstrebte Ziel ist die Erklimmung. Dies ist ein Beispiel für ein (elementares) Projekt, welches in der Theorie gerne zum Vergleich herangezogen wird.

Ein Projekt kann also im großen und ganzen als ein Entwicklungsprozess aus folgenden Bestandteilen angesehen werden:

1. einem klaren **Auftrag**, der sich aus Inhalten, Niveaus, Phasen und Schwerpunkten zusammensetzt.
2. **Zielen** auf 3 Ebenen: Man unterscheidet *Projektziele*(Projektebene), strategische Ziele (Strategieebene) und Ziele auf der Zielebene. Projektziele sind Ziele, die vordergründing mit dem Erreichen des Auftragsziels, also der Fertigstellung des Produktes oder der Ankunft an einem Ziel zusammenhängen. Bei strategichen Zielen handelt es sich um individuelle Ziele der Projektteilnehmer, die auf eine Verbesserung der Selbstständigkeit abzielen. An den Aufgaben, denen sich Projektteilnehmer in Zukunft stellen wollen, richten sich Ziele auf der Zielebene.
3. einem **Zeitrahmen**, den es (je nach Strenge) einzuhalten gilt.
4. **Arbeitsstrategien**, die sich aus Organisationsstrukturen ableiten, und eine Beziehung zwischen Theorie und Praxis herstellen.
5. unterschiedlichen **Arbeitsorten**.
6. eine **Projektkultur**, welche sich mit der Zeit entwickelt, und sich aufs Arbeitsklima auswirkt.

7. verschieden(st)en **Persönlichkeiten**, die sich in Alter, Bildungsgang und Voraussetzungen unterscheiden.

8. einem (dynamischen) **Beziehungsnetzt** unter den Mitwirkenden.

9. einer möglichst ausgewogenen Mischung aus **Intuition** und Verstand, als auch einer **Motivation**, die als Motor für intuitive und verstandsgelenkte Handlungen dient.

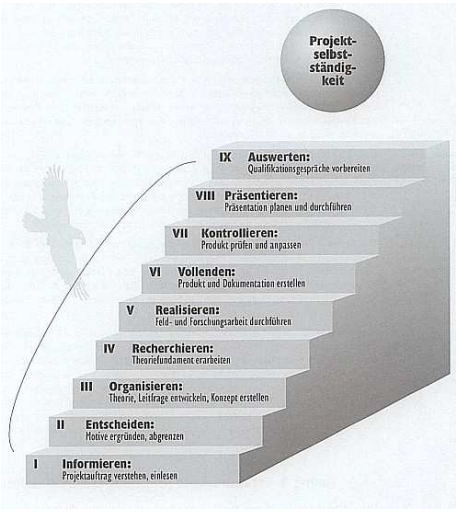
...und wer macht Projekte?

In die Passform „Projekt“ nach obiger Definition kann man also eine Menge von Aktivitäten im Alltag, als auch in der Unternehmens-, sowie Wissenschaftswelt einpassen. Für uns gilt es aber in erster Linie Aktivitäten nicht im Nachhinein als Projekte zu entlarven, sondern an Projekten zu partizipieren; Projekte zu verwirklichen.

Konzentrieren wir uns also auf Projekte im Sinne von Wirtschaft, Bildung und Forschung. Des weiteren gehen wir von Projekten aus, die sich zeitlich durchaus über Monate, als auch Jahre erstrecken können, sowie von mehreren Personen in Wechselwirkung bearbeitet und beeinflusst werden.

Als Mitwirkender bei einem Projekt nimmt man meist eine Rolle ein (*Projektrolle*), die sich zwischen dem unerfahrenen *Projektnovizen* und der Position des *Projektmeisters* befindet. Dabei wird hier vorwiegend der Erfahrungsstand, in Bezug auf bereits erfahrene Projekte, für die Einordnung herangezogen. Novizen sind dementsprechend Mitwirkende mit niedrigem oder noch keinem Erfahrungsstand. Meister, im Gegensatz dazu, blicken bereits auf eine beträchtliche Anzahl von erfolgreich bewältigter Projekten zurück. Meister werden in der Regel von Novizen für anspruchsvolle Projekte als Betreuer (*Projektcoaches*) herangezogen. Projektcoaches müssen keine Projektleiter sein. Bei ihnen handelt es sich in den meisten Fällen um Experten auf Teildisziplinen, deren sich eine auf Erfolg ausgerichtete Projektarbeit bedienen muss.

Die 9 Phasen (Projekttreppe)



Die 9 Projektphasen bilden „die Schritte der Entwicklung in Projekten“. Sie dienen der allgemeinen Orientierung während der Arbeit an einem Projekt. Die Grafik 10.2.2 gibt Aufschluss darüber, welche Phasen relevant sind, und in welcher Abfolge sie zum Projektziel, der obersten Treppe, führen. Diese Verbildlichung abstrahiert von den Variablen der Dauer einer Phase, der Schwerpunktlegung und (natürlich) den Inhalten, die im Projekt behandelt werden. Es gilt diese Treppenstufen in einer „linear-dynamischen Abfolge“ [KK01] zu bewältigen. Damit ist gemeint, dass man die Konzentration auf die aktuelle Phase legt, jedoch mit Blick auf die nächst niedrigere(n), bzw. höhere(n) Phase(n) arbeitet, und dabei das Ziel, das Treppenende, nicht aus den Augen verliert.

Zur Vermittlung eines allgemeinen Überblicks werden die Projektphasen anhand ihrer Arbeitsschwerpunkte beschrieben:

Informieren beginnt schon vor dem eigentlichen Projekt mit dem „herauspicken von *Projektrosinen*“ [KK01]. Ein Thema zu wählen, das zu einem passt, ist eine wichtige Voraussetzung für ein erfolgreiches Projekt (Motivation, Spassfaktor). Im *Projektauftrag* werden die Rahmenbedingungen gesteckt Diese sind gründlich zu studieren um frühzeitig Missverstädnissen vorzubeugen. Es gilt objektiv den Auftrag zu analysieren, und subjektiv zu erkennen wo Freiräume zum Einbringen der Persönlichkeit schaffbar sind. Allgemein ist dies die Phase in der man sich einen Überblick vermittelt.

Entscheiden tut, wer „nach Prüfen, Vergleichen oder kurzem Besinnen in einem Entschluss seine Wahl auf etwas festlegt“ [KK01]. Nachdem diese Themenwahl getroffen wurde, muss ein Themenrahmen im Team entwickelt werden, der möglichst „Motivation, Wünsche und Zielvorstellungen “ der Teilhabenden vereint, jedoch auf einen machbar, nicht zu breiten, Bereich eingrenzt, so dass ein klarer Rahmen erkennbar wird. Es ist

wichtig die Balance zu finden zwischen „Wünsch- und Machbarem, Vision und Realität, Voraussetzungen und geforderten Leistungen“ [KK01], Auftrag und wahrgenommenen Freiräumen.

Organisieren fordert zuallererst einmal ein *Konzept*(*Projektkonzept*), welches von der Projektleitung in Konsens mit dem Auftraggeber erstellt werden muss, und aus dem klar hervorgeht wie die Ziele verstanden und umgesetzt werden sollen. Erste Literaturrecherche und Klärung von Schlüsselbegriffen muss dabei betrieben werden. Dafür muss eine präzise Leitfrage formuliert werden, die mit der Umsetzung (Wo?, Wie?, Warum? usw. ...) des Projektes zusammenhängen. Je präziser die Leitfrage, desto genauer kann bei der Umsetzung des Konzeptes (hin zum Produkt) gearbeitet werden, so dass der Produktinhalt letztenendes auch mit dem Projektziel übereinstimmt.

Recherchieren stammt linguistisch vom französischen Wort „rechercher“ ab, und bedeutet soviel wie „aufdecken, herausfinden, ermitteln“. Informationsquellen müssen erschlossen und das Wissen aus bereits bekannten Quellen vertieft werden. Der Unterschied zwischen Fremdleistung von Eigenleistung ist hier zentral. Durch Selbst- und Fremdbewertungen des aktuellen, das Projekt betreffenden, Informationsstandes können Verständnisfehler und Recherchedefizite aufgedeckt und behoben werden.

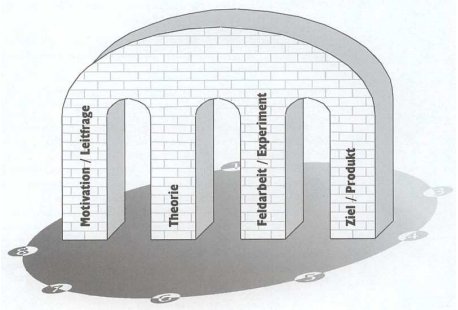


Abbildung 10.1: Projekttreppe

Realisieren bedeutet nun endlich „Die Brücke zwischen Theorie und Praxis “ [KK01] zu bauen. Leitfrage und Konzept (*Organisieren*) führen anhand der theoretischen Grundlagen (*Recherchieren*) zur Realisierung. Diese kann von Wissenschaftsfeld zu Wissenschaftsfeld stark varriieren (Interview, Experiment, Komponisation usw. ...). Große Aufmerksamkeit ist hier gefragt: Ist der Zeitrahmen einhaltbar ; integriere ich mich richtig; stehen die Ziele der 3 Ebenen (Projekt-, Strategie-, Zielebene) weiterhin im Vordergrund? Die Projektbrücke in Abbildung 10.2.2 hilft dem Fragenden dabei, das Hauptsächliche nicht aus den Augen zu verlieren.

Vollenden heisst Fertigstellung vom Produkt, als auch Anfertigung eines Projektberichts. Die aus dem vorigen Schritt (*Realisieren*) gesammelten Erfahrungen nehmen nochmal Einfluss auf das Produkt, welches jetzt zum großen Teil *vollendet* wird. Den Projektrichtlinien entsprechend, wird eine umfangreiche *Projektdokumentation* erstellt. Von der Leitfrage an, bis hin zum Produkt, wird hier der Werdegang des Projektes schriftlich widergegeben. Sie beinhaltet eine Stellungnahme des bzw. der Autoren und sollte, dem Zielpublikum angepasst, leserfreundlich gestaltet werden. „Jedes Projekt besteht somit aus einem Projektbericht... und dem eigentlichen Produkt ... .“ [KK01]

Kontrollieren des Produktes auf Tauglichkeit im Ernstfall ist unausweichlich zur Qualitätssicherung. Testläufe können noch ein letztes Mal auf Probleme oder Fehler hinweisen. Ehrliche Meinungen von Aussenstehenden darüber, ob die Komplexität und der Wirkungsgrad des Produktes und der Projektdokumentation dem Anspruch genügen, könnnen eingefahren werden. Die letzte Chance Anpassungen vorzunehmen sollte wahrgenommen werden. Ziel beim kontrollieren ist es das angepeilte Projektniveau zu erreichen.

Präsentieren meint, „etwas darbieten, der Öffentlichkeit vorstellen“ [KK01]. Eine Präsentation, die den Spagat zwischen fesselnder Darbietung und der seriösen Vermittlung des Inhalts schafft, ist der Schlüssel zum erfolgreichen Vermarkten seines Produktes. Das Interesse des Publikums durch provokante Fragen wecken, es so zu Aha-Effekten verleiten und die Wichtigkeit des Produktes klar und ohne Widersprüche zu vermitteln, das ist was einen guten Präsentator ausmacht. Weiterführende Informationen können dem Kapitel „Präsentieren mit LaTeX?“ entnommen werden.

Auswerten bildet den Abschluss eines Projektes. Ein ehrliches Auseinandersetzen mit dem Erwarteten und dem Erlangten, in Hinblick auf die 3 zu unterscheidenden Ebenen (Projekt-, Strategie- und Zielebene), ist tragend für zukünftige Projekte. Sind vielleicht Qualifikationen erlangt worden, die so nicht zu erwarten waren, oder haben Voraussetzungen zum Umsetzen von Aufgaben gefehlt? Fremd- und Selbstbewertung (z.B. anhand eines Bewertungsrasters) und klärende, abrundende Qualifikationsgespräche in kleinen Gruppen, geben darüber Aufschluss. Je nach Projektniveau wird die Auswertung allgemein oder sehr spezifisch anhand von Bewertungen vieler Teildiszipline des Projektes

(z.B. der Projekttreppen) erstellt.

10.2.3 Geniale Projekte, Schritt für Schritt entwickeln

Mit der Projekttreppe und ihren Stufen ist der statische Teil unsres projektdidaktischen Modells erklärt worden. Für anspruchsvolle, fortgeschrittene Projekte reicht allein dieses Sachwissen, also das Wissen auf der Projektebene, nicht aus.

Im folgenden werden wir noch die Ebenen 2 und 3 des projektdidaktischen Werkes anhand ihrer Ideologien und Schwerpunkte vorstellen.

Die Fragewörter, die die Orientierung der 3 Kapitel ausmachen:

- Das **Was?**(*Projektebene*),
- das **Wie?**(*Strategieebene*)und
- das **Wohin?**(*Zielebene*)

werden differenziert Betrachtet.

Strategieebene

Der Weg zur Selbständigkeit steht im Mittelpunkt dieses Kapitels. Talente müssen erkannt, Mankos entdeckt und wegtrainiert werden. Durch Zuhilfenahme der Kompetenzenkugel in Abbildung 10.2 kann ein jeder sein eigenes Profil erstellen (*Persönlichkeitsprofilbildung*), um über die Arbeit an an diesem, eine möglichst runde *Kugelgenialität* zu erlangen. Nach dem Motto „Probleme sind Lernchancen“ geht man diese dann über strukturierte Fragekataloge an.

Wie Selbstständigkeit über Projekte mit Projektzielsetzungen auf dieser Ebene erlangt werden kann, wird anhand von 9 Schlüsselaspekten(siehe Kompetenzenkugel) behandelt.

Zielebene

Wie durch gezielte Projektwahl und Training von Talenten nicht nur die Selbständigkeit verbessert, sondern darauf beruhend auch die Persönlichkeit gezielt erweitert werden

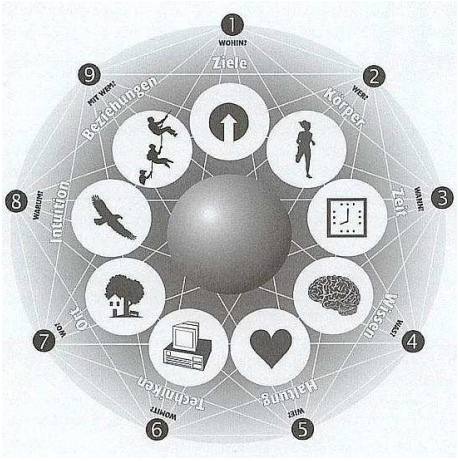


Abbildung 10.2: Kompetenzenkugel

kann, darüber wird in diesem Kapitel Aufschluss gegeben.

9 mögliche Profilierungsziele werden behandelt (z.B. Tatkraft, Sicherheit, Ausstrahlung). Durcharbeiten der Kapitel und Training der dort notwendigen Talente soll den Probanden seinem angestrebten Ziel näher bringen.

Im Gegensatz zu den Zielen auf der Strategieebene wird gefragt *„ Wo will ich mit meinen Projekttalenten hin?“*, nicht *„Wie kann ich weniger entwickelte Projekttalente erweitern?“*.

10.3 Wissenschaftliche Projekte

Was hat denn jetzt so ein Projektaufbau mit den Wissenschaften zu tun? Oder welche Schritte sind in bestimmten Wissenschaften wichtiger als andere?

Wenn wir uns jetzt wieder die drei Wissenschaften nehmen, auf die ich schon näher eingegangen bin:

In der Naturwissenschaft kann es wichtig sein, Versuche selber durchgeführt zu haben. Wichtig könnte auch sein, dass man gut mit der Mathematik vertraut ist.

In den Strukturwissenschaften, vorallem in der Informatik, sollte man Wert dadrauf legen wie man die Ausarbeitung gestalltet. Wenn ein Projekt zum Beispiel beinhaltet

ein bestimmtes Programm zu schreiben, sollte es nicht einfach nur funktionieren, sondern auch das „wie“ es geschrieben wurde, ist wichtig.

In der Geisteswissenschaft sollte vorallem Wert auf die Formulierung von Texten bzw des Vortrags geachtet werden. Denn durch manche Doppeldeutigkeit von Begriffen kann es schnell zu Missverständnissen kommen.

10.4 LaTeX vs. Word

Ein Vergleich von Word und LaTeX, was ist besser geeignet für welche Dinge.

10.4.1 Word: pro und contra

+ Einfachere Bedienbarkeit und Korrekturmöglichkeiten
+ leicht zu erlernen
+ Trennung von logischer Struktur und Layout möglich (Formatvorlagen)
+ Erstellen eines neuen Layouts einfach
+ Integrierte Rechtschreibprüfung
- Textsatz / Layout oft nicht befriedigend
- Binärformat, das ungewollte Informationen enthalten kann
- Zusammenspiel mit anderen Programmen des Betriebssystem

10.4.2 LaTeX: pro und contra

+ eine Art gräuliche Schrift die das lesen angenehmer macht [Müh03]
+ viele Dokumentklassen verfügbar
+ Benutzung vorgegebener Dokumentklassen einfach
+ Textsatz und Layout in Buchdruck-Qualität
+ unabhängig vom Betriebssystem

-/+ Trennung von logischer Struktur und Layout wir verlangt

- LaTeXist weniger gut geeignet für ein innovatives Layout

- Einstieg schwierig, komplizierte Fehlermeldung

10.4.3 Fazit

Man muss schauen wo man seine Schwerpunkte setzt. Hat man für sein Projekt nur wenige Tage Zeit und keiner aus der Gruppe kennt sich mit LaTeX aus, kann es sinnvoll sein Word oder ein ähnliches Schreibprogramm zu verwenden.

Ist das Ziel des Projekts aber zum Beispiel ein Buch oder ähnliches, ist es wohl angebracht LaTeX zu benutzen.

Kapitel 11

Mehrsprachige Texte in LaTeX

Erik Flick, Maurice Ossenbrink

11.1 Das Babel-Package

11.1.1 Motivation

LaTeX ist ursprünglich vor allem ein amerikanisches Projekt und darauf zugeschnitten, englischsprachige Texte umzusetzen. Man möchte LaTeX aber auch für weitere Sprachen einsetzen; Buchstaben können zwar ohne Weiteres mit Akzenten versehen werden, aber echt fremdsprachiger Schriftsatz mit korrekter Silbentrennung und Berücksichtigung typographischer Eigenheiten erfordern ein spezielles Package, wie auch Überschriften in anderen Sprachen. Babel vereinigt die früheren Einzelpackages und ist durch seinen Aufbau mit Modulen erweiterbar.

11.1.2 Was Babel bietet

Babel eignet sich, um für LaTeX auch andere Sprachen verfügbar zu machen, die mit Varianten des lateinischen oder verwandter Alphabete geschrieben werden. Zudem erlaubt es, mehrsprachige Dokumente zu verfassen.

Sonderzeichen und diacritics sind schon erzeugbar; Babel erleichtert dies auch durch Abkürzungen; aber Babel bietet mehr.

- Ziemlich viele Sprachen:
- Übersetzungen der Standard-Überschriften
- Silbentrennung und Typographie
- Mehrere Sprachen in einem Text (Wechsel im Text, anderssprachige Umgebungen)
- Mit allen Dokumentklassen, inputenc, fontenc kompatibel

Sprachmodule

Die sprachspezifischen Definitionen sind in Modulen festgelegt, somit ist Babel leicht erweiterbar.

Es existieren Module für eine breite Auswahl an Sprachen. Hier eine unvollständige Aufzählung: *afrikaans, bahasa, breton, catalan, croatian, czech, danish, dutch, english, american, british, esperanto, estonian, finnish, francais, galician, german, ngerman, magyar, irish, italian, lowersorbian, norsk, nynorsk, polish, portuguese, brazilian, romanian, russian, scottish, spanish, slovak, slovene, swedish, turkish, uppersorbian, welsh.*

Gelegentlich gibt es für eine Sprache mehrere Module, die weiterentwickelt werden, aber Unterschiede aufweisen.

Am Beispiel von “germanb”:

Shorthands

Babel definiert Abkürzungen für häufig verwendete nationale Sonderbuchstaben, um Texte besser eingebbar und leserlicher zu machen: Statt

```
{\"{a}}
```

beispielsweise

```
"a
```

, um den Buchstaben ä zu erzeugen.

Typographie

Babel legt Trennungsregeln (hyphenation rules) zur Silbentrennung am Zeilenumbruch fest; die deutschen Module enthalten zudem Definitionen für Buchstabenkombinationen, die bei der Trennung anders aussehen:

```
"c
```

für ein “ck”, das zum “k-k” wird.

Übersetzungen für die LaTeX-Standardüberschriften

LaTeX-Überschriften, solche wie “Chapter” oder auch die Datumsangabe, die in LaTeX in Englisch gehalten sind, werden durch Babel auch übersetzt: 25 août 2005 - 25. August 2005

11.1.3 Anwendung von Babel im Dokument

Vorsicht: Babel wird ständig weiterentwickelt, und die Befehle bewahren oft von einer Version zur Nächsten nicht das gleiche Verhalten. Hier trotzdem die Wichtigsten:

Das Package einbinden:

```
\usepackage[german]{babel}
```

oder:

```
\documentclass[german]{article}
\usepackage{babel}
```

damit wird die Information über die gerade geladenen Sprachen auch anderen Packages, die danach geladen werden, etwa inputenc, verfügbar gemacht.

Mehrere Sprachen in einem Text

Man kann Babel so einbinden:

```
\usepackage[francais,german]{babel}
```

So werden alle angegebenen Module geladen, das letzte steht für die Sprache, die aktiv ist.

Es gibt mehrere Möglichkeiten, im Text die Sprache zu wechseln:

Entweder eine Umstellung, die ab der Stelle, an der das Kommando vorkommt, gilt; dabei wird *nicht* auf Umgebungen geachtet:

```
\selectlanguage{sprache}
```

Dieses Kommando ruft mehrere Makros auf die alle Parameter (Trennungsregeln, Überschriften, Abkürzungen...) umstellen.

```
\begin{otherlanguage}{sprache}
```

...

```
\end{otherlanguage}
```

Umgebung: Wechselt die Sprache.

```
\foreignlanguage{sprache}{...}
```

Textabschnitt in anderer Sprache gesetzt - Silbentrennung und Specials - möglichst nur für die Länge eines Paragraphen benutzen.

```
\otherlanguage*
```

Läßt die Überschriften beim Alten, wechselt nur die "Specials"

```
\iflanguage{sprache}{...}{...}
```

Führt den ersten Block auf, wenn die Sprache ausgewählt ist, sonst den Zweiten.

11.2 Typographische Eigenarten

In verschiedenen Sprachen gelten teilweise auch verschiedene typographische Konventionen. Häufig sind diese aber auch nicht genau festgelegt und damit eine Frage der persönlichen Präferenz. Ich möchte hier einige der typographischen Optionen vorstellen hauptsächlich mit Blick auf englischsprachige und deutschsprachige Texte.

11.2.1 Absätze

Der Standard in LaTeX sind eingezogene Absätze ohne eine freie Zeile zwischen den einzelnen Absätzen. In deutschen Dokumenten wird es aber teilweise eher bevorzugt, wenn Absätze durch Leerzeilen gekennzeichnet sind. Ein Einzug ist in diesem Fall nicht mehr notwendig.

Um dies für das gesamte Dokument zu aktivieren hat man mehrere Möglichkeiten. Zum Beispiel gibt es das Parskip Package, das dies automatisch aktiviert. Es ist aber auch möglich, manuell Werte für Abstand und Einzug zu setzen. In der Präambel sieht das dann folgendermaßen aus:

```
\usepackage{parskip}
oder
\setlength{\parindent}{0pt}
\setlength{\parskip}\medskipamount
```

Bei der Option für parskip hat man die Möglichkeit den Abstand absolut anzugeben oder auch in relativen Angaben wie small- med- oder bigskipamount.

Beispiel:

> Das ganze sieht dann aus wie hier. Emuliert durch die quote Umgebung.
>
> Diese benutzt nämlich im Gegensatz zur quotation Umgebung keine Einzüge.

Es ist auch möglich Einzüge manuell zu unterdrücken bzw. zu erzeugen. Das geschieht durch die Befehle:

```
\indent
\noindent
```

Manchmal kann es ganz sinnvoll sein, auch in Dokumenten mit Einzügen, zwischen einzelnen Absätzen einen gewissen Abstand zu setzen, um visuell deutlich zu machen, dass an diesem Punkt ein neuer Gedanke folgt. Dafür gibt es je nach Größe des gewünschten Abstandes die folgenden Befehle:

```
\smallskip
\medskip
\bigskip
```

Beispiel:

> Manchmal ist es ganz gut, wenn man kleine Abstände lässt, die einem sagen:
>
> „Hier beginnt etwas Neues!“
>
> Aber auch ganz neue Ideen Verlangen manchmal nach noch
>
> Grösseren Abständen.

11.2.2 Frenchspacing

Im Englischen Textsatz ist es üblich hinter Satzzeichen einen größeren Abstand zu lassen als bei einem normalen Leerzeichen.

Frenchspacing unterdrückt dies und verwendet lediglich normale Abstände. In der german Umgebung von Babel ist Frenchspacing automatisch aktiviert. Es kann aber auch manuell de- bzw. reaktiviert werden mit:

```
\frenchspacing
\nonfrenchspacing
```

> Hier ist kein Frenchspacing. Hier ist kein Frenchspacing. Hier ist kein Frenchspacing. Hier ist kein Frenchspacing.
>
> Hier ist Frenchspacing. Hier ist Frenchspacing. Hier ist Frenchspacing. Hier ist Frenchspacing.

11.2.3 Anführungszeichen

Normale Schreibmaschinen-Anführungszeichen werden im Textsatz eigentlich nicht benutzt. Sie können aber manuell erzeugt werden mit

```
\dq
```

Beispiel:

> Guck dir mal die "Gänsefüßchen" an.

Normalerweise werden im Deutschen die deutschen Anführungszeichen verwendet. Erzeugt werden sie folgendermaßen.

```
"‘ und "’
oder
\glqq und \grqq
```

Für Zitat-in-Zitat o.ä. benötigt man die deutschen einfachen Anführungszeichen:

```
\glq und \grq
```

Beispiel:

> Er erzählte: „Und dann rief sie ‚Hilfe!‘ und ich rettete sie.“

Besonders in Romanen, Erzählungen usw. werden im Deutschen auch französische Anführungzeichen verwendet.

Allerdings werden diese im Französischen genau andersherum benutzt, weswegen man im Deutschen links und rechts vertauschen muss, um das gewünschte Ergebnis zu erhalten.

```
"> und "<
oder
\frqq und \flqq
```

Für einfache Anführungszeichen verwendet man:

```
\frq und \flq
```

Beispiel:

> Er erzählte: »Und dann rief sie ›Hilfe!‹ und ich rettete sie.«

Im Englischen werden wieder andere Anführungszeichen benutzt. In der doppelten Version werden sie so erzeugt:

```
`` und ''
```

Für einfache Anführungszeichen:

```
` und '
```

Ob man im Englischen für normale Zitate doppelte oder einfache Anführungszeichen benutzt hängt hauptsächlich vom persönlichen Geschmack ab. Tendenziell werden im Britischen eher einfache im Amerikanischen eher doppelte Anführungszeichen verwendet.

Eine weitere Besonderheit im Englischen ist, dass bei einem Zitat, das über mehrere Absätze geht, ein führendes Anführungszeichen vor jedem neuen Absatz gesetzt werden sollte.

Beispiel:

> ‘The Babel fish,’ said The Hitchhiker’s Guide to the Galaxy quietly, ‘is small yellow and leechlike, and probably the oddest thing in the universe
>
> ‘...
>
> ‘The argument goes something like this: “I refuse to prove that I exist,” said God, “for proof denies faith, and without faith I am nothing.”
>
> ‘ “But”, says Man, “the Babel fish is a dead giveaway isn’t it? It could not have evolved by chance. It proves you exist, and so therefore, by your own arguments, you don’t. QED.”
>
> ‘ “Oh Dear,” says God, “I hadn’t thought of that,” and promptly vanishes in a puff of logic. ...’

11.3 Unicode

Unicode provides a unique number for every character,
no matter what the platform,
no matter what the program,
no matter what the language.

11.3.1 Zeichensatzprobleme bei Mehrsprachigkeit

Zueinander inkompatible Zeichensätze sorgen für Chaos

ASCII, der Zeichensatzstandard Bisherige Erweiterungs-Zeichensätze zu ASCII, das nur das lateinische Alphabet enthielt (was größtenteils ausreichend ist, um englischen Text darzustellen) zeichneten sich dadurch aus, daß sie jeweils nur eine begrenzte Anzahl von Sprachen unterstützten, richtig eingestellt werden mußten und nicht ohne weitere Angaben erkannt wurden, und kaum innerhalb eines Dokuments kombinierbar waren. Zudem gab es oftmals konkurrierende "Codepages" bzw. Zeichencodierungen für die gleichen Sonderbuchstaben. Mehrere Sprachen in einem Text einzusetzen, war problematisch, da der Text mit einer bestimmten Codierung zu lesen war, die unter Umständen nicht alle benötigten Sonderzeichen (auf einmal) unterstützte.

11.3.2 Vereinheitlichung

Heute ist Unicode (www.unicode.org[1]) *der* anerkannte Standard für Zeichencodierung. Es soll alle zur Zeit gebrauchten (und auch alle historischen) Schriftsysteme umfassen, und davon ist es nicht mehr weit entfernt; Erweiterung um fehlende Schriften geschieht laufend, wobei einmal festgeschriebene Codierungen nicht mehr revidiert werden, um Kompatibilitätsprobleme auszuschließen. Vorgesehen ist Platz für ca. eine Million Zeichen (Version 4.0 definiert 96.447). Bei der Anwendung soll man nicht mehr auf Zeichensätze achten müssen, weil hier jedem Zeichen eine eindeutige Nummer zugewiesen ist.

[1] Das Unicode Consortium, setzt sich zusammen aus den größten Computer- und Softwareherstellern, Regierungen...

11.3.3 Mehr als ein Zeichensatz

Unicode legt nicht nur die Zeichen fest, sondern z.B. auch Codierungen wie UTF-8, Algorithmen, nach denen beispielsweise bidirektionaler (teils in einer von rechts nach links geschriebener) Text zu behandeln ist, Such- und Sortierfunktionen...

11.3.4 Repräsentation von Unicode-Zeichen

Unicode-Zeichen nehmen in der Repräsentation, in der sie zur Verarbeitung vorgehalten werden (UTF-32), 4 bytes ein; abgespeichert werden sie für gewöhnlich im UTF-8 (Unicode Transformation Format) abgespeichert, welches 1 byte (beginnend mit 0) für die Codes 0-127 benutzt und somit zu Texten, die nur aus ASCII-Zeichen bestehen, kompatibel ist; 2 bytes für die Codes 128-2047, 3 bytes für Codes bis 65535 und 4 bytes für alle Übrigen:

Bereich	Bytes	Codes (binär)
0-127	1	0xxxxxx
128-2047	2	110xxxxx 10xxxxxx
2048-65535	3	1110xxxx 10xxxxxx 10xxxxxx
65536-	4	11110xxx 10xxxxxx 10xxxxxx 10xxxxxx

11.3.5 Zur Anwendung

Unicode selbst legt lediglich die Codierung fest; um die Schriften auch tatsächlich darstellen zu können, braucht man eine Schriftart, die die entsprechenden Unicode-Bereiche abdecken muß.

11.3.6 Anwendung in LaTeX

Unicode-Text können wir auch direkt eingeben. (Das ganze Latex-Dokument in UTF8 einzulesen, sollte zukünftig auch möglich sein.)

```
\usepackage{ucs}
\usepackage[utf8]{inputenc}
```

(ucs mit anzugeben, scheint nicht mehr nötig). Weitere Anwendungshinweise finden sich im Kapitel über ostasiatische Sprachen.

11.4 Ostasiatische Sprachen

Die großen ostasiatischen Sprachen haben alle ihre eigenen Zeichensätze, die nicht kompatibel miteinander sind. Zwar enthalten diese Zeichensätze in der Regel auch alle ASCII Zeichen, möchte man aber in dem selben Dokument auch Umlaute verwenden oder gar eine zweite ostasiatische Sprache, kommt man um Unicode für die Kodierung nicht herum. [2]

Dafür gibt es das Package für Unicode Support UCS, das sofern installiert in der Präambel geladen wird.

```
\usepackage{ucs}
```

Unicode besitzt außerdem noch den Vorteil, dass es systemunabhängig ist.

Für den Sprachsupport an sich gibt es das CJK Package, das - wie der Name schon andeutet - Support für Chinesisch, Japanisch und Koreanisch liefert. Deklariert wird es mit:

```
\usepackage[encapsulated]{CJK}
```

Zusätzlich liefert das CJK Package noch Support für so genannte Rubies (im Japanischen: Furigana), also die Möglichkeit, die Lesung eines Zeichens über diesem anzuzeigen. Darüberhinaus gibt es noch die Möglichkeit die traditionelle vertikale Schriftweise zu verwenden. Da vertikale Schrift von LaTeX eigentlich nicht unterstützt wird, dreht CJK bei diesem Modus einfach alle Zeichen um 90 Grad. Deklariert werden diese Funktionen mit den zusätzlichen Packages:

```
\usepackage[overlap, CJK]{ruby}
\usepackage{CJKvert}
```

[2] Theoretisch ist es natürlich auch möglich für die verschiedenen Sprachen verschiedene Kodierungen zu verwenden. Das bietet sich aber in den meisten Fällen nur an, wenn man nicht vorhat, mehrere Sprachen in einem Editor zu bearbeiten.

11.4.1 Die CJK Umgebung

Um die ostasiatischen Sprachen in einem Dokument einzubinden, muss man zuerst eine CJK Umgebung öffnen.

```
\begin{CJK}{UTF8}{cyberbit}
\CJKhorz

\end{CJK}}
```

Zusätzlich zur CJK Umgebung muss noch die Kodierung, als in diesem Fall Unicode (UTF8) und die Familie des benutzten Fonts angegeben werden. Ich verwende in diesem Beispiel die frei erhältliche TrueType Unicode Schriftart Cyberbit. Der `\CJKhorz` Befehl ist nur notwendig, wenn das Paket für vertikale Schrift geladen ist, da ansonsten der Text automatisch vertikal gesetzt wird.

Am einfachsten ist es, wenn man sich dafür ein Makro definiert. Also z.B.

```
\newcommand{\cjk}[1]{\begin{CJK}{UTF8}{cyberbit}\CJKhorz#1\end{CJK}}
```

11.4.2 Koreanisch

Als erstes Beispiel, hier ein kurzer koreanischer Text in der Silbenschrift Hangul. Mit dem oben definierten Makro funktioniert das so:

```
\cjk{hier steht der koreanische Text}
```

한글배우기가어렵지않아요.

Auf Deutsch: „Hangul lernen ist nicht schwer“

11.4.3 Japanisch mit Furigana

Japanische Eingabe funktioniert genauso wie in dem koreanischen Beispiel oben. Für die Furigana Eingabe benötigt man den Befehl `\ruby`. Er wird wie folgt verwendet:

```
\ruby{zu lesendes Zeichen}{Lesung}
```

z.B. ergibt `\ruby{日本語}{にほんご}` folgendes: 日本語 (にほんご)
Das selbe funktioniert auch mit lateinischen Lettern:
これは日本語 (Japanisch) ですか？ -え？！Deutsch (ドイツ語) でしょう！

11.4.4 Japanisch in vertikaler Schreibweise

Japanisch in vertikaler Schriftweise in ein horizontales Dokument einzubinden gestaltet sich etwas schwieriger, da im vertikal Modus des CJK Package lediglich die Zeichen um 90 gedreht werden. Das Ganze sieht dann so aus:

みなさん今日は！

Um den vertikalen Text mit dem Rest des Dokuments in Einklag zu bringen bietet es sich an, den Text in eine Textbox zu setzen und diese wiederum um 90 Grad zu drehen. dafür benutzt man z.B. das Rotating Package:

```
\usepackage{rotating}
```

Als Beispiel hier ein berühmtes Haiku von Basho:

```
\begin{turn}{-90}
\parbox{3cm}{
\cjkvert{hier steht das Haiku}}
\end{turn}
```

古池や
蛙とびこむ
水の音

In der deutschen Übersetzung:

Der alte Teich.
Ein Frosch springt hinein -
das Geräusch des Wassers

Kapitel 12

Fonts in LaTeX

Yvonne Küstermann, Natalja Fiodorova

Zuerst wird eine kurze Einführung in das Formatieren von Texten gegeben. Anschließend wird der Frage nachgegangen, für welche Zwecke Formatierungen und verschiedene Schriftarten von Nöten sind. Ein weiteres Thema sind die in LaTeX vorhandenen Symbole. Die passenden Fonts und Symbolfonts zu wählen, um zu einem abgerundeten Layout zu kommen, ist Thema dieses Abschnittes.

Wichtig hierfür ist Wissen über die grundlegenden Zusammenhänge in LaTeX. Aus diesem Grund wird in den darauffolgenden Abschnitten auf Kodierung von Schriftarten und auf ihre Definitionen eingegangen. Dabei werden die Programmiersprache MetaFont und die Schriftformate PostScript und TrueType vorgestellt.

Eine besondere Gewichtung wird auf PostScript-Fonts gelegt, die im letzten Teil dieses Kapitels behandelt werden.

12.1 Formatieren

Für Schriftarten sind folgende fünf Merkmale zu beachten:

- Kodierung
- Familie
- Serie
- Gestalt

- Größe

Auf diese fünf Merkmale und ihre Variation wird in diesem Kapitel eingegangen. Im nächsten Abschnitt werden drei dieser Mermale behandelt, nämlich Familie, Serie und Gestalt.

Befehl – Umgebung – Deklaration

Wichtig bei Schriftarten ist die Unterscheidung zwischen Serifenschriften, serifenlosen Schriften und Maschinenschriften. Die Voreinstellung in einem LaTeX-Dokument ist die Schrift Computer Modern Roman. Wie 'Roman' schon besagt, handelt es sich hierbei um eine Serifenschrift. Um nun in Maschinenschrift zu wechseln, gibt es eine Reihe verschiedener Möglichkeiten. Anhand dieses Beispiels wird die allgemeine Weise in LaTeX zu formatieren demonstriert.

Unterschieden wird in LaTeX zwischen den Konzepten Befehl, Deklaration und Umgebung, wobei sich die letzten beiden sehr ähneln.

Befehl

Ein Befehl nimmt meist ein oder aber auch mehrere Argumente. Mittels eines Befehls setze ich meinen Text folgendermaßen in Maschinenschrift:

```
Das tt in \verb|\texttt{}|
steht f"ur \texttt{Teletype}.
```

Das tt in `\texttt{}` steht für `Teletype`.

LaTeX-Befehle sind nicht möglichst kurz, sondern tendieren eher dazu möglichst aussagekräftig zu sein. Aus diesem Grund beginnen alle Formatierunsbefehl, bis auf den `\emph{}`-Befehl mit 'text'. Häufig bietet es sich daher an, eigene, kürzere Befehlsnamen zu definieren.

Für längere Textabschnitte empfiehlt sich die Verwendung von Befehlen aber nicht, da sie den zu formatierenden Text ja als ihr Argument haben.

Umgebung

Einen längeren Text setze ich in Maschinenschrift indem ich ihn in eine Maschinenschrifttumgebung schreibe.

```
\begin{ttfamily}
Hier steht jetzt
ein langer Text ...
\end{ttfamily}
```

```
Hier steht jetzt ein langer Text ...
```

Deklaration

Die dritte Möglichkeit sind Deklarationen. Durch die Deklaration `\ttfamily` wird LaTeX veranlasst alles weitere in Maschinenschrift zu setzen. Die gilt jeweils bis zum Ende einer Umgebung. Die Deklarationen lassen sich auch mit einem Klammerpaar umschließen, so dass sie genau innerhalb dieses Bereiches gelten. Ansonsten entfaltet die Deklarationen ihre Wirkung innerhalb einer Umgebung in der sie steht.

```
{\ttfamily Maschinenschrift}
\\ttfamily Maschinenschrift
```

```
Maschinenschrift
```

In Tabelle 12.1 sind alle Formatierungen aufgelistet. Umgebung und Deklaration weisen jeweils die gleiche Syntax auf. Bemerkenswert ist hier, dass Teil der Namen das Fontmerkmal ist, das geändert wird. Um bei dem Beispiel mit der Maschinenschrift zu bleiben: hier ist das Merkmal 'Familie' Teil des Kommandos.

12.1.1 Schriftgröße

Im Gegensatz Schriftfamilie, -serie und -gestalt wird bei der Schriftgröße nur mit Deklarationen gearbeitet.

Möchte ich klein schreiben, kann ich das folgendermaßen machen:

```
{\scriptsize kleine Schrift}
```

kleine Schrift

In Tabelle 12.2 sind die zur Verfügung stehenden Größen aufgelistet.

Tabelle 12.1: Formatieren

`\textrm{`*text*`}`	`\rmfamily`	Antiqua
`\textsf{`*text*`}`	`\sffamily`	Serifenlose
`\texttt{`*text*`}`	`\ttfamily`	`Maschinenschrift`
`\textmd{`*text*`}`	`\mdseries`	normal
`\textbf{`*text*`}`	`\bfseries`	**fett, breiter laufend**
`\textup{`*text*`}`	`\upshape`	aufrecht
`\textsl{`*text*`}`	`\slshape`	*geneigt*
`\textit{`*text*`}`	`\itshape`	*kursiv*
`\textsc{`*text*`}`	`\scshape`	Kapitälchen
`\textnormal{`*text*`}`	`\normalfont`	Die Grundschrift des Dokuments

12.1.2 Fortgeschrittenes Formatieren

Die LaTeX-Befehle lassen sich selbstverständlich auch kombinieren, so dass beispielsweise Schriftfamilie und Schriftgröße gleichzeitig geändert werden und das Ganze dann auch noch fett erscheint. Hierzu folgendes Beispiel:

```
Wunderbare Kombinationen
{\ssfamily\bfseries\large
von Sansserif, fett und gro"s}
lassen sich kreieren.
```

Wunderbare Kombinationen **von Sansserif, fett und groß** lassen sich kreieren.

Wie schon erwähnt sind die Befehlsnamen in LaTeX eher sprechend gewählt als kurz. Aus diesem Grund empfiehlt es sich, sich eigene Befehle und Umgebungen zu definieren. Zudem lässt sich so der logische Aufbau des Textes besser darstellen. So lässt sich der Befehl für das Kursivsetzen mit einem Namen versehen, der zeigt, was das Kursivsetzen im Dokument bewirken soll. Sollen alle Autorennamen im Text kursiv gesetzt werden könnte ein befehl namens `\autor` definiert werden. Entscheide ich mich anschließend für eine andere Formatierung, muss außerdem nur diese eine Befehlsdefinition geändert werden.

Tabelle 12.2: Schriftgrößen

`\tiny`	winzig kleine Schrift
`\scriptsize`	sehr kleine Schrift (wie Indizes)
`\footnotesize`	kleine Schrift (wie Fußnoten)
`\small`	kleine Schrift
`\normalsize`	normale Schrift
`\large`	große Schrift
`\Large`	größere Schrift
`\LARGE`	sehr große Schrift
`\huge`	riesig groß
`\Huge`	gigantisch

Die folgenden beiden Zeilen setzen ihr Argument jeweils fett.

```
\newcommand{\meinFett}[1]{\textbf{#1}}

\meinFett{Diese Befehle sind echt FETT.}

\newcommand{\perfektesFett}[1]{{\nomalfont\bfseries #1}}

{\perfektesFett Diese Befehle sind echt FETT.}
```

Die erste Möglichkeit definiert einfach einen neuen Namen anstelle von **textbf**. Die zweite schaltet erst auf Normalfont um und setzt dann ihr Argument fett. Überraschend sind hierbei auf den ersten Blick die doppelten Klammern. Das erste Klammerpaar kommt von dem newcommand-Befehl. Das zweite Klammerpaar umschließt die Deklaration, so dass ihre Auswirkung auf den Teil innerhalb der Klammern begrenzt ist.

12.2 Schriftarten

Die Standardeinstellung in LaTeX ist Computer Modern Roman. Um dies zu dem serifenlosen Pendant zu verändern, wird der Default geändert. Dies geschiet mittels:
`\renewcommand\familydefault{cmsf}`
Bekannt sein muss jeweils lediglich das Kürzel für die Schriftart. Einige Beispiele sind: Es lassen sich auch viele weiter Defaults ändern, so z.B. rmdefault, welcher für die Serifenschrift verwendet wird oder ttdefault, der festlegt, welche Maschinenschrift LaTeX standardmäßig verwendet.

Die oben geschilderte Vorgehensweise ändert das ganze Dokument von Serifenschrift zu serifenloser Schrift oder zu einer etwas ausgefalleneren Schrift wie Computer Modern Funny Roman. Was aber bezwecken wir mit unterschiedlichen Schriftarten? Zwei Situationsbeschreibungen machen die Aufgabe von Schriften deutlich.

- Ich möchte bestimmte Teile meines Textes in eine andere Schriftart setzen.
- Ich möchte für mein gesamtes Dokument ein anderes Layout, denn 8ich kann die LaTeX-Voreinstellungen nicht mehr sehen.

Schriftarten als Mittel zum Formatieren

In der ersten Situation, wenn also bestimmte Textteile in einer anderen Schriftart erscheinen sollen, ist es sinnvol sich einen Befehl zu definieren.

```
\newcommand{\willAndereSchrift}[1]%
{{\fontfamily{txr}\selectfont #1}}
{\willAndereSchrift Dies ist die Schrift Times.}
```

Dann sieht das Ergebnis folgendermaßen aus: Dies ist die Schrift Times.

Wie an dem Beispiel zu sehen, wird in eine andere Schriftart gewechselt, indem `\fontfamily` mit dem Kürzel der entsprechenden Schriftart angegeben wird. Anschließend wählt LaTeX die spezifizierte Schrift mit `\selectfont`.

Mittels dieser Methode lassen sich alle fünf Merkmale von Fonts manipulieren. Dazu werden die folgenden Befehle benötigt:

```
\fontencoding{}
\fontfamily{}
\fontseries{}
\fontshape{}
\fontsize{}{}
```

`\fontsize` nimmt als einziges zwei Argumente. Hierbei wird als erstes die Zeilenhöhe angegeben und als zweites Argument die Schriftgröße in Punkt.

Änderung des Designs des Dokuments

Möchte ich allerdings das Layout von Grund auf verändern, ist die beste Möglichkeit ein entsprechendes Package zu laden.
Einige Beispiel hierfür:

```
\usepackage{cmbright}
\usepackage{helvit}
\usepackage{luximono}
\usepackage{fourier}
\usepackage{txfonts}
\usepackage{pxfonts}
```

12.2.1 Wie finde ich die passende Schriftart?

Nun ist es ja schön Packages und die Namen von Schriftarten zu kennen, nur hilft das alleine noch nicht weiter. Wie sehe ich denn nun, ob mir die Schriftart gefällt? Wie finde ich das passende Layout für mein Dokument?

nfssfont.tex

Die Idealvorstellung ist, dass ich einige Seiten Papier habe und dann den entsprechenden Stil aussuchen kann. Kann ich dem Coputer denn nicht sagen: „Hei, zeig mir mal alles, was du so drauf hast!“? Eine solche Lösung ist mir leider nicht bekannt, aber zumindestens gibt es ein Programm, mit dem ich einzelne Schriftarten testen kann. Hierzu schicke

ich die Datei nfssfont.tex durch LaTeX und werde dann interaktiv nach der Schriftart, die ich testen möchte, gefragt. Anschließend erwartet das Programm eine Angabe über den Output. Möglich sind `\text` und `\table`. In ersterem Fall wird ein voreingestellter Beispieltext ausgegeben, im zweiten Fall erhalte ich eine Tabelle mit allen vorliegenden Zeichen. Ich beschließe meine Eingbe mit `\bye`.

```
michel@host:~ $ latex nfssfont

*******************************************
* NFSS font test program version <v2.0e>
*
* Follow the instructions
*******************************************

Name of the font to test = xipasb10
Now type a test command (\help for help):)
*\text
*\table
*\bye
[1]
Output written on nfssfont.dvi (1 page, 12260 bytes).
Transcript written on nfssfont.log.
michel@host:~ $
```

Literaturempfehlung

Eine Reihe von verschiedenen Packages sind im LaTeX-Companion zu finden, so dass es möglich ist, sich anzusehen wie verschiedene Packages dann auf dem Papier tatsächlich wirken:
LaTeX-Companion [MG04], Kapitel 8.8.3: „A collection of math font set-ups“.

Hilft weder nfssfont.tex noch der LaTeX-Companion richtig weiter, muss eben das Package oder die Schriftart selbst per Hand ausprobiert werden ...

12.3 Fontkodierung

Die Standardkodierung in LaTeX ist OT1. Dies ist die 'alte' Kodierung in der die Zeichen mit nur 7-bit repräsentiert werden. Daraus resultieren 128 Zeichen. Diese Anzahl ist für die englische Sprache ausreichend, nicht jedoch für zusätzliche Zeichen oder diakritisch Zeichen, wie sie in anderen Sprachen vorkommen. Dadurch ist es schwer möglich Wörter mit Umlauten zu trennen. Zudem wird die Typographische Qualität beeinträchtigt. Ein Buchstabe mit Akzent wird dann als Kombination aus dem jeweiligen Buchstaben und dem Akzent erzeugt. Dies harmoniert nicht immer. Die T1-Kodierung dagegen kodiert jedes Zeichen mit 8 Bit. Nun ist es möglich in mehr als 30 Sprachen mit lateinischer Schrift zu schreiben. Um die T1-Kodierung zu nutzen, ist folgendes notwendig:
`\usepackage[T1]{fontenc}`

Im Großen und Ganzen sind die resultierenden Fonts bei OT1 und T1 ähnlich. In kleinen aber manchmal doch interessanten Details unterscheiden sich die Ergebnisse dann aber doch. Das ß wurde für die T1-Kodierung neu designed und die Schriftgrößen unterscheiden sich leicht.

Mit der OT1
Kodierung: ß.
Mit der T1
Kodierung: ß.

Es existieren eine Reihe weiterer Kodierungen für unterschiedliche Zwecke und unterschiedliche Sprachen. Beispielsweise bezeichnet OT2 die kyrillische Schrift. Meist ist es aber gar nicht nötig dies manuell zu ändern, da das Sprachpaket diese Aufgabe schon übernimmt.

Die Nützlichkeit von Kodierungen lässt sich anhand des Phonetischen Alphabetes demonstrieren. Dies ist die Kodierung T3 und sie wird automatisch bei der Nutzung des Packages tipa geladen:

```
\usepackage{tipa}
```

'tipa' steht für 'International Phonetic Alphabet' und dieses Alphabet ist eine international gabräuchliche Lautschrift. Auf die Kodierung wird mittels des Befehls `\textipa{ }`

ASCII	:	;								
TIPA	ː	ˈ								
ASCII	0	1	2	3	4	5	6	7	8	9
TIPA	ʉ	ɨ	ʌ	ɜ	ɥ	ɐ	ɒ	ɤ	ɵ	ɘ
ASCII	@	A	B	C	D	E	F	G	H	I
TIPA	ə	ɑ	β	ɕ	ð	ɛ	ɸ	ɣ	ɦ	ɪ
ASCII	J	K	L	M	N	O	P	Q	R	S
TIPA	ʝ	ʁ	ʎ	ɱ	ŋ	ɔ	ʔ	ʕ	ɾ	ʃ
ASCII	T	U	V	W	X	Y	Z	|		
TIPA	θ	ʊ	ʋ	ɯ	χ	ʏ	ʒ	ǀ		

Tabelle 12.3: TIPA shortcut characters

zugegriffen. Bemerkenswert ist, wie viele Zeichen in T3 mit nur einem 'nomalen' Zeichen wiedergegeben werden können. Dies ist in Tabelle 12.3 zu sehen. Erreicht wurde dies, indem die Großbuchstaben und Zahlen umdefiniert wurden zu phonetischen Zeichen. Daduch ist auch der Quelltext noch sehr gut lesbar, obwohl der Editor die phonetischen Zeichen natürlich nicht darstellen kann. Dies soll illustrieren, dass eine logisch durchdachte Kodierung die Arbeit sehr erleichtern kann. Selbstverständlich ist es möglich auch eine eigen zu definieren. Diese beginnen mit dem Buchsatben L und es folgen zwei weiter Zeichen.

Nun nehme ich es mir vor, in einem beliebigen in lateinischer Schrift gesetzten Text einzelne russische Wörter oder ganze Absätze einzufügen. Die Dokumentation von Babel erklärt, wie man diese Aufgabe unter Verwendung von dem sogenannten `T2A`-Zeichensatz löst. Dabei soll der kyrillische Text aber direkt eingegeben werden. Geht es mir nun um das „gelegentliche Schreiben“ an einer westlich ausgelegten Tastatur, so habe ich entweder das Kyrillische „blind“ einzugeben oder nach einer weiteren akzeptablen Lösung zu suchen.

Also, ich habe bei der Verfolgung meines Zieles folgende Nebenbedingungen:

- Ich soll einen Font mit dem kyrillischen Zeichensatz herausfinden, der im Aussehen zur hauptsächlich benutzten Schrift passt,
- Das Verfahren zur Eingabe der kyrillischen Zeichen mit ASCII-Umschrift muss

existieren,

- Die russischen Trennmuster müssen für das Funktionieren der Silbertrennung im LaTeX-Format vorhanden sein,
- Das Babel-System wird zur Umschaltung zwischen verschiedenen Trennmustern und Schriften verwendet.

Die ersten beiden Nebenbedingungen werden erfüllt, wenn man die sogenannte OT2-Kodierung verwendet. Der Nachteil hierbei ist, dass die automatische Silbentrennung nicht mehr tadellos funktioniert. Das liegt an den Eingabeligaturen und daran, dass OT2 nicht alle möglichen Kombinationen aus Buchstaben und diakritischen Zeichen „fertig“ enthält. Letzteres Problem kennen wir auch vom OT1-Encoding.

Wir wollen Babel zur sprachabhängigen Umschaltung zwischen Trennmustern und Fonts verwenden. Das Paket muss mit der Option `russian` geladen werden, und als letzte Option ist die im Dokument als Voreinstellung gewünschte Sprache, z.B. `german`, anzugeben. Wichtig ist, dass `babel` *nach* `fontenc` geladen wird: Denn dann „weiß“ Babel, dass es die OT2-Kodierung und nicht seine Voreinstellung `T2A` für die russischen Textteile verwenden soll:

```
\usepackage[OT2,T1]{fontenc}
\usepackage[russian,german]{babel}
```

Zum Setzen russischer Textteile innerhalb des Dokumentes dienen dann der Befehl `\foreignlanguage` oder die Umgebung `\otherlanguage`. Z. B. ergibt

```
\foreignlanguage{russian}{Student}
```

die Ausgabe Student, also das russische Wort für „Student“.

✎ Für eine detaillierte Beschreibung des Verfahrens und eine Auflistung der Umschrift aller möglichen kyrillischen Zeichen siehe [Sch03].

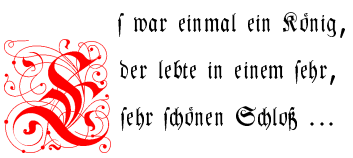
12.4 Symbole

Aufgrund des NFSS ist es relativ einfach, weitere Zeichensätze für LaTeX bereitzustellen. Es müssen jedenfalls zwei Voraussetzungen erfüllt sein:

- ❏ Die dazugehörigen Fontdateien müssen auf dem System installiert sein und
- ❏ Das entsprechende Paket ist nach dem `\documentclass` anzugeben.

Um beispielsweise die altdeutschen Schriftsätze zu verwenden, reicht es aus, das Modul `yfonts` zu importieren, indem man es mit dem `\usepackage`-Befehl und der entsprechenden Option in der Präambel einbindet:

```
\usepackage{color}
\usepackage[german]{babel}
\usepackage[varumlaut]{yfonts}
⋮
\frakfamily
\fraklines
\yinipar{
\color{red}E}s war einmal
ein K"onig, der lebte
in einem sehr,
sehr sch"onen Schlo"s ...
```



Sucht man eventuell nach alternativen Fonts für mathematische Formeln, findet man bei den Euler Fonts sehr schöne Zeichen mit der handschriftlichen Note. Es stehen insgesamt drei Mathematikalphabete zur Verfügung:

- ❒ Euler Roman Alphabet kann man über den Befehl `\mathcal`,
- ❒ Euler Script Alphabet über `\mathscr`, und dann beide unter Verwendung des Paketes `eucal` ansprechen,
- ❒ Euler Fraktur Alphabet lädt man entsprechend mit dem Alphabetbefehl `\mathfrak` und dem Paket `eufrak`.

Möchte man Textsymbole einbinden, findet man diese im `TS1`-Zeichensatz. Verwendet man aber schon die `T1`-Schriften, stellt das Paket `textcomp` die Lösung bereit. Es enthält viele verschiedene Zeichen, wie Währungssymbole (einschl. des Euro), Copyright und

Copyleft, alle Symbolzeichen aus ISO-8859-1 (Latin-1), Mediävalziffern[1] und viele andere mehr.

Das folgende, einfache Beispiel demonstriert die Funktionsweise des `textcomp`-Paketes:

```
\usepackage{textcomp}
⋮
Alexander S. Puschkin,
\textborn {26.05.1799};
\textdied {29.01.1837}
gilt als bedeutendster
russischer Dichter.
```

Alexander S. Puschkin,
⋆26.05.1799; †29.01.1837
gilt als bedeutendster russischer Dichter.

Einige der Textsymbole sind auch ohne das Paket `textcomp` verfügbar, dann aber nicht immer in einem zum laufenden Schriftsatz passenden Stil.

✎ Ausführliche Beschreibungen zu den `textcomp`-, `euler`- und `yfonts`-Paketen sind in [MG97, Kapitel 7.5.] zu finden.

12.5 Fontdefinitionen

Dieser Abschnitt ist den verschiedenen Formaten, in denen Fonts definiert werden, gewidmet, d.h. den Dateien, in denen die Form der Zeichen beschrieben ist.

12.5.1 MetaFont

Das älteste (von Donald Knuth geschaffene), aber dennoch raffinierteste Format von allen, insbesondere im Hinblick auf typografisches Design. In MetaFont legt man Linienzüge durch Punkte und durch diese Punkte festgelegte Spline-Kurven fest, und zieht diese mit einem vorher definierten „Pinsel“ nach. Durch die Form und die Lage des verwendeten Pinsels können mit denselben Kurven sehr unterschiedliche Zeichenformen erzeugt werden. Nachdem das Zeichen aus einzelnen Kurvenzügen definiert und mit

[1] nach oben und unten versetzte Ziffern, wie z. B. 0123456789

unterschiedlichen Pinseln gezeichnet wurde, rastert MetaFont die dadurch eingefärbten Flächenteile und erzeugt daraus einen Font in Form einer Bitmap.

Da MetaFont zur Programmierung einer solchen Font-Beschreibung eine Programmiersprache definiert, kann ein Font-Designer einer Font-Beschreibung beliebig viele Parameter mitgeben, die die Form und Ausmaße der einzelnen Zeichen der erzeugten Schrift in unterschiedlichen Richtungen verändern. Wegen dieser Möglichkeit, aus einer einzigen Beschreibung durch Wahl verschiedener Parameter viele verschiedene Schriften erzeugen zu können, nennt man diese Font-Beschreibungen auch „Meta-Font“.

Seine Hauptnachteile sind, dass nur TeX es versteht und vor allem die Notwendigkeit, vorher die Dateien in der korrekten Auflösung zu erzeugen (mit dem damit verbundenen Verbrauch an Speicherplatz).

12.5.2 PostScript

Neben dieser von MetaFont verwendeten Beschreibung eines Fonts in Form von mit Pinseln gezeichneten Spline-Kurven kann man die Form eines Zeichens auch durch die Angabe des Umrisses der zu schwärzenden Fläche beschreiben. Auch hier werden wiederum Spline-Kurven zur Beschreibung der Umrisse eines Zeichens verwendet. Diese Beschreibung in Form eines sogenannten „Outline“ wird beispielsweise in PostScript-Schriften verwendet.

PostScript ist eine vollständige, auf Forth basierende Programmiersprache, die unter diesem Namen seit 1984 von der Firma Adobe entwickelt wird. PostScript:

- stellt Methoden zur Verfügung, das Erscheinungsbild inkl. Text, Linien und Graphiken zu beschreiben,
- unterstützt viele Datentypen (`integer`, `real`, `boolean`, `array`, `String`, Files und Kontrollstrukturen),
- hat sich zu einem Standard in der Druckindustrie entwickelt, wird aber immer mehr von PDF (*portable document format*) verdrängt.
- ist geräte- und auflösungsunabhängig.

Im Folgenden soll das Prinzip von PostScript erklärt werden. Die Ausgabe der Druckseite erfolgt in 2 Schritten:

① die Anwendung

- ➥ erzeugt auf einen Ausgabebefehl hin ein PostScript-Programm, das Format und Gestaltung des Dokumentes beschreibt und
- ➥ sendet es an das Ausgabegerät;

② das Gerät

- ➥ interpretiert das Programm und
- ➥ setzt es in die Steuerbefehle zur Ausgabe des Dokumentes um.

Seine Vorteile sind seine hervorragende Qualität, die Speicherplatzersparnis und die fast grenzenlose Veränderbarkeit.

12.5.3 TrueType

Das Format wurde von Apple geschaffen, um technisch unabhängig von den PostScript-fonts von Adobe zu sein, aber seine Qualität stellte sich als unterlegen heraus, und das Ganze endete in einem Fiasko. Wenn Microsoft es nicht gekauft hätte, wäre es wahrscheinlich heute verschwunden. Sein zweiter Nachteil ist, dass es heutzutage nur in der nicht-professionellen Windows-Welt ernstgenommen wird, und daher nahmen es nur wenige TEX -Implementationen an, erwähnenswerte Ausnahme darunter - pdftex.

Die TrueType–Schriften werden nach dem Prinzip einer Vektorgrafik aus Konturen gebaut. Erst bei der Ausgabe werden die Konturen mit Pixeln gefüllt. Der daraus resultierende Vorteil ist die verlustfreie Skalierbarkeit.

12.6 PostScript-Fonts

In diesem Kapitel wird kurz auf die Möglichkeiten von NFSS und des virtuellen Fontmechanismus eingegangen, die die Verwendung von PostScript-Fonts ermöglichen. Mithilfe von dvips-Treiber werden dann die im Drucker residente oder (von Festplatte oder Diskette) ladbare PostScript-Fonts mit LATEX verwendet.

12.6.1 dvips – PostScript-Treiber

Der von Tomas Rokicki entwickelte PostScript-Treiber dvips unterstützt bei:

Graphiken:

- das Einfügen,
- automatische Skalierung sowie
- Positionierung;

Zeichensätzen:

- automatische Generierung der fehlenden Fonts mit Metafont (soweit auf dem System vorhanden),
- Unterstützung der virtuellen Fonts, wodurch die Verwendung von PostScript-Fonts mit TeX erlaubt wird.

12.6.2 Virtuelle Fonts

Ein virtueller Font ist (wie der Name schon sagt) eine Schrift, die keine real (in Form einer Bitmap) existierenden Zeichen enthält. Stattdessen wird in einem virtuellen Font auf andere Schriften und die dort enthaltene Zeichen verwiesen. Durch diesen „Trick“ kann man beispielsweise:

- die Kodierung der Zeichen in einer Schrift umdefinieren oder
- eine Schrift aus Zeichen mehrerer Schriften zusammensetzen, um so z. B. einzelne Zeichen durch andere zu ersetzen.

Darüber hinaus kann man in virtuellen Fonts auch

- einzelne Zeichen aus mehreren Zeichen zusammensetzen, indem man bsp. fertig akzentuierte Zeichen aus dem entsprechenden Akzent- und Basiszeichen bildet.

Eine weitere Anwendung für virtuelle Fonts ist

- ❖ die Verwendung einer Ersatzschrift für eine nicht verfügbare Schrift, die dieser möglichst nahekommt und dieselben TEX -Font-Metric-Informationen besitzt.

Eine `vf`-Datei enthält die Definition eines solchen virtuellen Fonts in binärer, sehr kompakter Form. Dieser `vf`-Datei muss eine `tfm`-Datei zugeordnet sein, in der TEX die „TEX Font Metric“ für diese Schrift finden kann. TEX „weiß“ nichts von virtuellen Fonts. TEX liest keine vf-Dateien, sondern benötigt nur die zugehörige `tfm`-Datei, aus der nicht ersichtlich ist, ob dies ein „realer“ oder ein virtueller Font ist. `vf`-Dateien werden nur von den Gerätetreibern verwendet.

12.6.3 PSNFSS-System

Das PSNFSS-System besteht aus einer langen Reihe von Paketen, die die Verwendung von PostScript-Fonts in LATEX 2_ε vereinfachen. Das PSNFSS-System verwendet die Berry-Nomenklatur. Dadurch wird eine einheitliche Identifizierung der Fonts auf verschiedenen Systemen erreicht.

Der eindeutige Name, den Karl Berry für Fonts vorgeschlagen hat, besteht aus 8 Zeichen (dank unseres alten Bekannten MS-DOS):

❶ Das erste Zeichen ist der Eigentümer der Schriftart: `p` (Adobe), `e` (Apple),
❷,❸ Die nächsten beiden bezeichnen die Schriftfamilie: `hv` (Helvetica), `tm` (Times), `pl` (Palatino),
❹ Das vierte Zeichen ist immer die Schriftstärke: `r` (Text), `b` (Fettdruck),
❺ Danach kommen die „Alternativen“: `i` (kursiv), `c` (Kapitälchen),
❻,❼ Hierauf kommt die Kodierung: `8a` (8-bit Adobe Standart), `7t` (7-bit Original TEX),
❽ Am Ende fügt man die Breite an: `n` (eng), `x` (weit). Wenn der Name mit `rr` endet, verkürzt man es auf ein `r`.

☞ Somit wäre beispielsweise `ptmr8t` – Times Schriftart von Adobe, Text, mit der 7-bit-Original-TEX -Kodierung.

Um die Funktionalität des dvips besser nachvollziehen zu können, betrachten wir ein kleines Beispiel:

Wie bekannt, besorgt dvips die Konversion des von TeX verwendeten Namens in den realen Fonts. Sagen wir, beispielsweise `ptmr8t`.
Wenn dvips in der `dvi`-Datei auf ein `tfm` dieses Fonts stößt, von dem eine `vf`-Datei vorhanden ist, ersetzt es dieses durch `tfm`, in diesem Fall `ptmr8t.tfm`. Das wird von der `vf`-Datei angezeigt und beginnt von vorn, nach einer `vf`-Datei zu suchen. Da es die `vf`-Datei nicht existiert, fährt es fort, indem es nachsieht, ob es für das `tfm` einen Eintrag in `psfonts.map` gibt. Dort findet es einen „echten" Font `ptmr8t`, den es PS Times-Roman nennt. Hätte letzteres auch fehlgeschlagen, hätte es schließlich nach einem `pk`-Font gesucht.

✎ Weitere Informationen zu MetaFont, Virtual Fonts und dvips sind über eine Suchmaschine bzw.
`www.dante.de/cgi-bin/ctan-index` zu finden.

12.6.4 PostScript Pi-Fonts

Zu guter Letzt möchte ich etwas über Pi-Fonts erzählen. Das sind die Fonts, die eine Ansammlung von Spezialzeichen enthalten, die man normalerweise in einem Textfont nicht findet. Sie sind unter Verwendung des pifont-Paketes, welcher auch Teil von PS-NFSS-System ist, verfügbar.

Die direkt verwendbare Zeichen des PostScript-Fonts ZapfDingBats sind mit dem `\ding`-Befehl auszuwählen. Der Parameter für den `\ding`-Befehl ist dann eine ganze Zahl, die gemäß des zu setzenden Zeichens anzugeben ist.

Zum Beispiel ergibt `\ding{37}`-Befehl einen ✍ als Ausgabe.

Möchte man eine spezielle `itemize`-Liste verwenden, tut man dieses mit der `\dinglist`-Umgebung.

Als eine weitere Möglichkeit steht der Befehl `\dingline` für das Füllen einer ganzen Zeile mit einem beliebigen ZapfDingBats-Zeichen zur Verfügung. Es sei an der Stelle noch erwähnt, dass ich während der Aufnahme dieses Artikels mehrmals die PostScript Pi-Fonts verwendet habe.

Literaturverzeichnis

[Div97] Div.: *dtv Lexikon*. F.A. Brockhaus GmbH und Deutscher Taschenbuch Verlag, 1997.

[Dow02] Downes, Michael: *Short Math Guide for LaTeX*, 2002.

[Fri03] Friedrich, Wolfgang G.: *Die Kunst zu präsentieren*. Springer-Verlag, 2003.

[Gou00] Goualard, Frédéric: *Manual for the prosper class, Version 1.0i (Documentation 1.6)*, 2000.

[GRM97] Goossens, Michel, Sebestian Rahtz Frank Mittelbach: *The LaTeX Graphics Companion*. Addison-Wesley, 1997.

[Han04a] Hansen, Thorsten: *The bibunits Package*, 2004. `http://tug.ctan.org/tex-archive/macros/latex/contrib/bibunits/bibunits.dtx`.

[Han04b] Hansen, Thorsten: *The multibib Package*, 2004. `http://tug.ctan.org/tex-archive/macros/latex/contrib/multibib/multibib.dtx`.

[Hei02] Heinz, C.: *The Listings Package*. Im Listings Paket: doc/latex/listings/listings.dvi, 2002.

[KK01] Kunz-Koch, C. M: *Geniale Projekte - Schritt für Schritt entwickeln*. orell füssli Verlag AG, 2001.

[KM04] Kohm, M. J.-U. Moraski: *Das KOMA-Script-Paket*. Im texmf-tree: doc/latex/koma-script/scrguide.pdf, 2004.

[Kop96] Kopka, Helmut: *LaTeX - Einführung*. Addison-Wesley, second , 1996.

[Lam86] LAMPORT, LESLIE: *LaTeX - User's Guide and Reference Manual.* Addison-Wesley, first , 1986.

[Los05] LOSEM, FREDERIC: *Indexerstellung mit Latex*, 2005. `http://www.informatik.hu-berlin.de/~losem/studium/ps2html.html#index`.

[MG97] M. GOOSENS, F. MITTELBACH, A. SAMARIN: *Der LaTeX-Begleiter.* Addison-Wesley, Bonn, 1997.

[MG04] M. GOOSSENS, F. MITTELBACH: *The LaTeX-Companion.* Addison-Wesley, Boston, 2004.

[Müh03] MÜHLICH, MATTHIAS: *Vergleich: Word und (pdf)LaTeX*, 2003. http://user.uni-frankfurt.de/ muehlich/tex/wordvslatex.html.

[N.N99] N.N.: *User's Guide for the amsmath Package*, 1999.

[NN03] NIEDERMAIR, ELKE MICHAEL NIEDERMAIR: *LaTeX – Das Praxisbuch.* Franzis, 2003.

[N.N04] N.N.: *Using the amsthm Package*, 2004.

[Pat88a] PATASHNIK, OREN: *BIBTeXing*, 1988. Im texmf-tree: \texmf\doc\bibtex\btxdoc.tex.

[Pat88b] PATASHNIK, OREN: *Designing BIBTeX Styles*, 1988. Im texmf-tree: \texmf\doc\bibtex \btxhak.tex.

[Pro99] PROJECT, THE LATEX 3: *LaTeX 2e for class and package writers.* http://www.latex-project.org/guides/clsguide.pdf, 1999.

[Rai] RAICHLE, BERND: *Tutorium: Einführung in die BIBTeX- Programmierung.* DANTE, Erlangen.

[Sch03] SCHMIDT, W.: *Kyrillisch für arme Leute: Setzen russischer Textpassagen mit LaTeX*, may 2003. http://home.vr-web.de/was/x/pmcyr.pdf.

[Ste05] STEIN, SEBASTIAN: *Eine kleine Latex Einführung*, 2005. `http://www.hpfsc.de/default.php?url=./latex/index.html`.

[Tan04] Tantau, Till: *User's Guide to the Beamer Class, Version 3.01*, 2004. http://latex-beamer.sourceforge.net.

[v1.04] v1.1, TDS Working Group: *The TeX Directory Structure*. http://www.tug.org/tds/, 2004.

[vdB99] Berg, Wim van den: *Autorität und Schmuck - Über die Fuktion des Zitates von der Antik bis zur Romantik*, 11–36. Amsterdam, 1999.

[Voß02] Voss, Herbert: *Die mathematischen Funktionen von Postscript*. 2002.

[Wik05] Wikipedia: *Wissenschaft*, 2005. www.wikipedia.de.

[Wil00] Wilson, David C.}: *The Utopia PDF Presentation Bundle, Version 1.22*, 2000.

[Zan] Zandt, Timothy Van: *PSTricks Documentation*. http://www.tug.org/applications/PSTricks/.

Index